

Divide and Conquer : Penerapannya dalam Kompresi Gambar dengan Metode Quadtree

Maheswara Bayu Kaindra (13523015) dan Bryan Ho (13523029)

13523015@std.stei.itb.ac.id; 13523029@std.stei.itb.ac.id

1 Latar Belakang

Dalam era digital saat ini, gambar dan video menjadi bagian tak terpisahkan dari kehidupan sehari-hari manusia. Seringkali, besarnya ukuran berkas gambar seringkali menjadi perhatian dan kendala bagi proses penyimpanan dan transmisi data, terutama bagi sistem yang memiliki keterbatasan ruang atau *bandwidth*. Oleh karena itu, pengembangan sistem kompresi yang efisien dan tetap menjaga kualitas gambar menjadi isu yang layak untuk diperbincangkan.

Salah satu pendekatan yang efektif untuk kompresi gambar adalah dengan struktur data *Quadtree*. Struktur data ini membagi area dua dimensi menjadi empat sub-area secara rekursif berdasarkan keseragaman karakteristik data di dalamnya, layaknya sebuah pohon dengan empat anak daun. Dalam konteks kompresi gambar, *Quadtree* digunakan untuk membagi citra menjadi blok-blok piksel yang dikelompokkan berdasarkan nilai warna dan intensitas pikselnya. Selain meningkatkan efisiensi ruang, pendekatan ini juga terbukti mampu mempertahankan informasi visual penting dalam gambar.

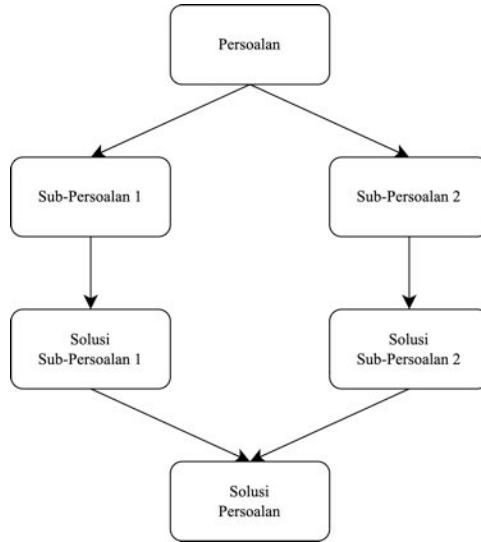
Tugas kecil ini bertujuan untuk mengaplikasikan *Divide and Conquer* melalui struktur data *Quadtree* dalam kompresi gambar dengan memanfaatkan berbagai metode pengukuran *error* seperti variansi, *mean absolute deviation*, *max pixel difference*, *entropy*, dan *structural similarity index*. Program yang dikembangkan diharapkan dapat menghasilkan sistem kompresi yang efisien dengan kualitas yang masih dapat diterima. Selain itu, diharapkan proyek ini menjadi pembelajaran aplikatif yang menerapkan Strategi Algoritma dalam kasus nyata.

2 Landasan Teori

Dalam proyek ini, kompresi gambar dilakukan dengan pendekatan algoritma *Divide and Conquer* yang menerapkan struktur data *Quadtree*. Untuk dapat memahami dasar dari metode yang diimplementasikan, diperlukan pemahaman mengenai dua aspek utama, yaitu *Divide and Conquer* dan konsep dasar kompresi gambar, khususnya dengan menerapkan struktur data *Quadtree*.

2.1 Divide and Conquer

Menurut Munir (2025), *Divide and Conquer* dibagi menjadi *divide*, *conquer*, dan *combine*. *Divide* berarti membagi persoalan menjadi beberapa sub-persoalan yang memiliki karakter serupa dengan persoalan asli, dengan ukuran yang lebih kecil. *Conquer* berarti menyelesaikan masing-masing sub-persoalan (bisa secara langsung atau rekursif). Setelah semua sub-persoalan diselesaikan, dilakukan proses *combine*, yaitu menggabungkan kembali masing-masing solusi sub-persoalan sehingga membentuk solusi persoalan semula.



Gambar 01 : Diagram Divide and Conquer

Dibuat dengan draw.io

Dalam *Divide and Conquer*, setiap sub-persoalan memiliki karakteristik yang sama dengan karakteristik persoalan semula, namun ukurannya lebih kecil. *Divide and Conquer* secara natural diimplementasikan dengan skema rekursif. Berikut merupakan skema umum algoritma *Divide and Conquer*.

```

procedure DIVIDEandCONQUER(input P : problem, n : integer)
Deklarasi
  r : integer

Algoritma
  if n ≤ n0 then
    SOLVE persoalan P yang berukuran n.
  else
    DIVIDE menjadi r sub-persoalan, P1, P2, ..., Pr dengan ukuran masing-masing n1, n2, ..., nr.
    for masing-masing P do
      DIVIDEandCONQUER(Pi, ni)
    endfor
  endif

```

Tabel 01 : Skema Umum Divide and Conquer

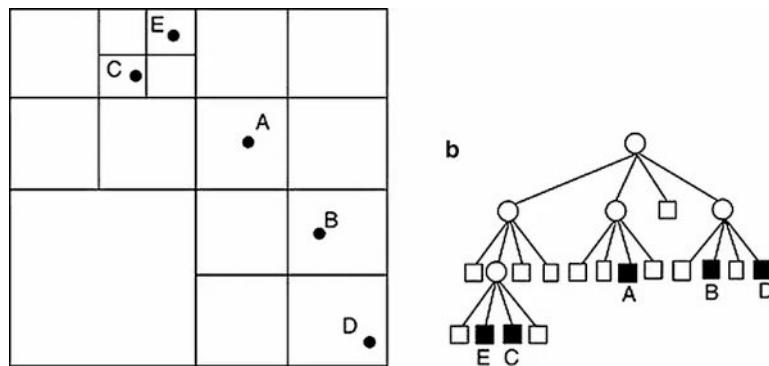
Secara kompleksitas algoritma, *Divide and Conquer* memiliki kompleksitas algoritma $T(n)$ yaitu sebagai berikut.

1. $g(n)$, untuk $n \leq n_0$
2. $T(n_1) + T(n_2) + \dots + T(n_r) + f(n)$, untuk $n > n_0$, di mana

$T(n)$ adalah kompleksitas waktu penyelesaian persoalan P yang berukuran n ; $g(n)$ adalah kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil; $f(n)$ adalah kompleksitas waktu untuk COMBINE solusi dari masing-masing sub-persoalan.

2.2 Metode Quadtree dalam Kompresi Gambar

Gambar dalam jumlah yang besar dapat menggunakan sangat banyak memori. Terkadang, saking banyaknya memori yang digunakan, semakin berat juga gambar tersebut untuk di *load*, sehingga terdapat kasus di mana gambar tidak dapat dilihat dalam skala penuh. Di situlah kompresi gambar dibutuhkan, salah satunya dengan *Quadtrees*. *Quadtree* merupakan sebuah struktur data berbentuk pohon di mana setiap simpulnya memiliki empat anak.



Gambar 02 : Ilustrasi Quadtree

Sumber : <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>

Dalam kompresi gambar, *Quadtree* bekerja secara rekursif dengan membagi kelompok pixel gambar menjadi empat bagian di mana setiap kelompok memiliki karakteristik yang serupa. Terdapat faktor error yang menentukan apakah suatu kelompok pixel perlu dibagi lagi menjadi empat atau tidak. *Quadtree* terdiri dari dua struktur, yaitu simpul dalam dan simpul daun.

1. Simpul dalam merepresentasikan blok-blok pixel yang belum seragam dan perlu dipecah lagi menjadi empat blok.
2. Simpul daun menunjukkan blok yang dianggap cukup seragam dan tidak perlu dibagi lagi.

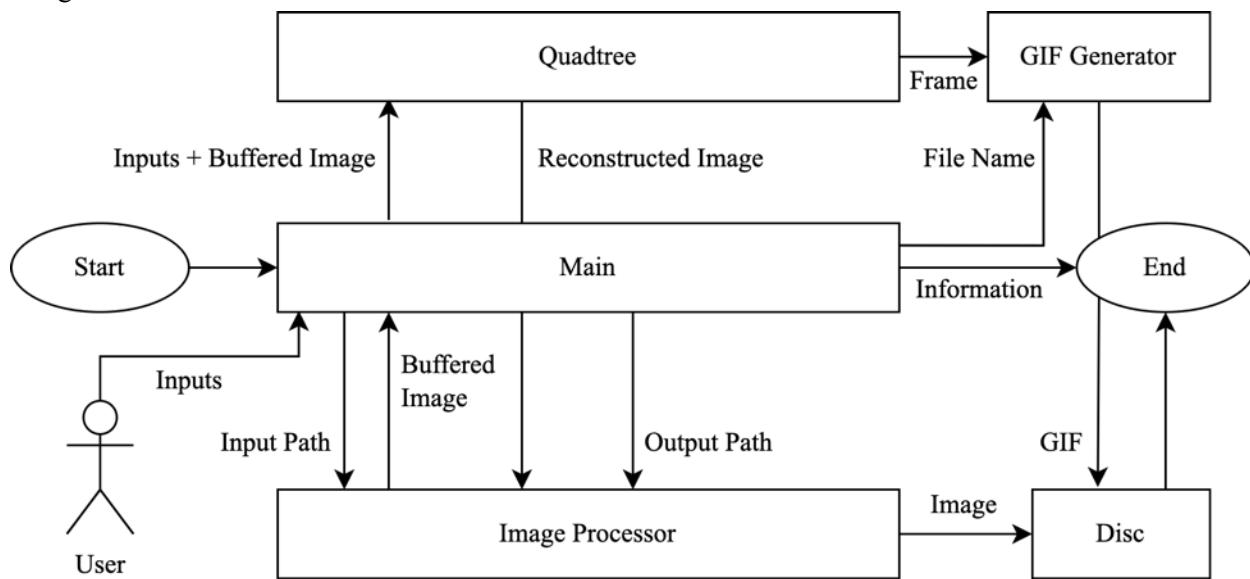
Setiap simpul daun menyimpan informasi seperti posisi koordinat, ukuran blok, serta rata-rata nilai warna blok dalam RGB. Dengan ini, informasi hanya disimpan pada simpul-simpul daun sehingga mengurangi redundansi data.

Metode *Quadtree* unggul karena tidak mengompres semua bagian gambar. Berbeda dengan metode kompresi konvensional yang mengompres seluruh bagian gambar (membuatnya blur secara rata), metode *quadtree* hanya mengompres bagian yang dianggap sudah seragam, namun tetap mempertahankan bagian yang seharusnya detail. Dengan ini, gambar akan lebih ringan tanpa kehilangan detail yang signifikan. Pendekatan ini sejalan dengan prinsip *lossy*, yaitu menghilangkan informasi yang tidak penting, namun mempertahankan yang memiliki peran signifikan.

3 Desain dan Penerapan Algoritma

3.1 Alur Umum

Secara umum, berikut merupakan alur pengolahan gambar menjadi gambar yang terkompresi. Program utama akan meminta pengguna untuk memasukkan alamat gambar absolut, metode pengukuran error, *threshold error*, ukuran blok minimum, dan alamat absolut untuk file output. Keseluruhan program akan memanggil satu sama lain untuk memproses gambar tersebut. Setelah gambar output selesai dibuat, pengguna diminta untuk memasukkan satu karakter ('y' atau 'n') sebagai pilihan untuk menyimpan proses kompresi menjadi sebuah file .gif. File .gif akan disimpan dengan nama yang sama dengan file output pada folder test. Penulis menggambarkan alur tersebut dalam sebuah *data flow table* sederhana sebagai berikut.



Gambar 03 : Ilustrasi interaksi antar struktur program.
Dibuat dengan draw.io

3.2 Desain Algoritma

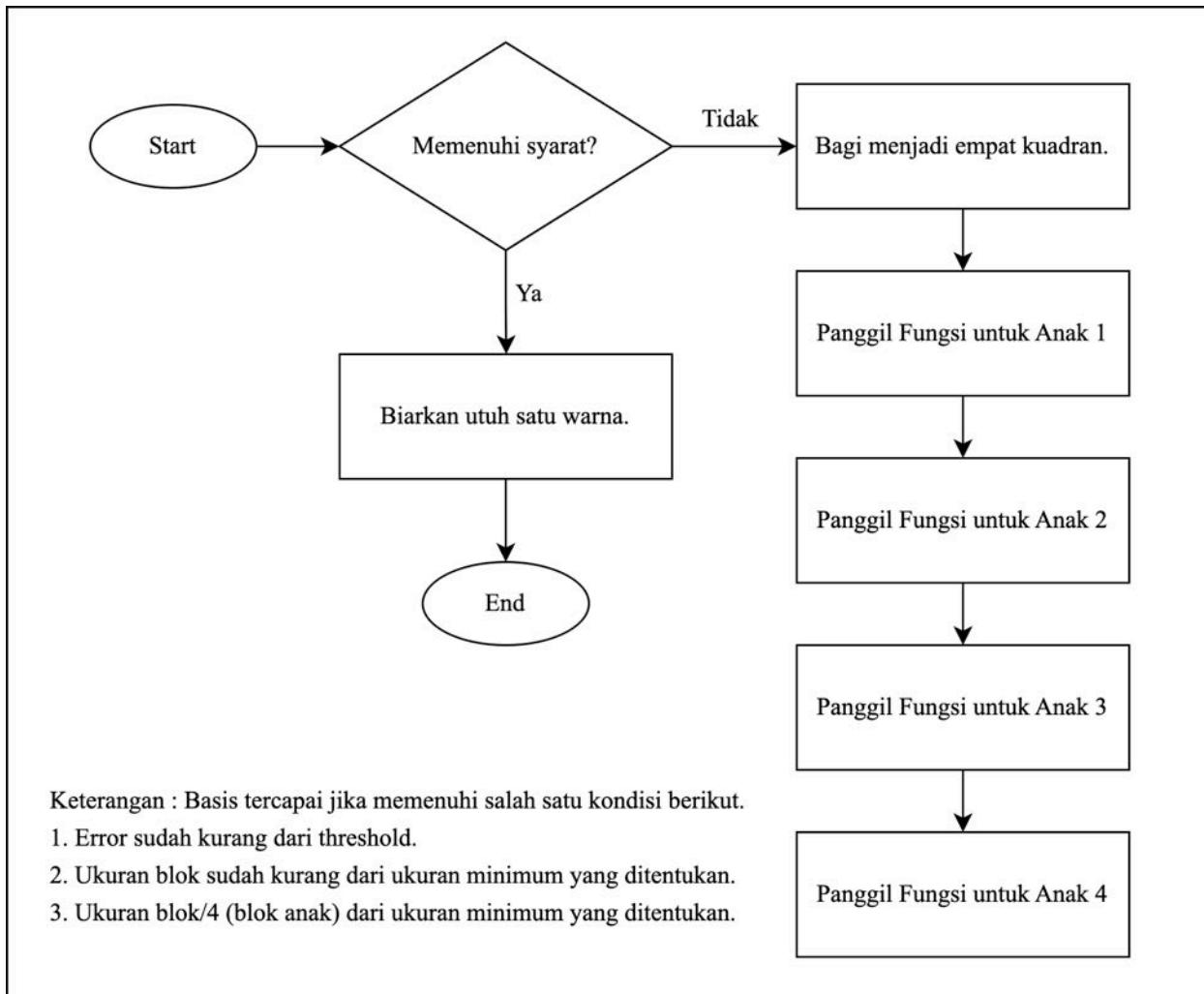
Secara umum, tidak ada yang terlalu spesial dari *Divide and Conquer* yang digunakan. Basis dari rekursi adalah membiarkan daerah dan mengisinya dengan warna sesuai rata-rata warna dalam format RGB, apabila error sudah dibawah threshold atau daerah induk atau daerah anak lebih kecil dari luas daerah minimum.

Apabila tidak memenuhi syarat, dilakukan pembagian daerah menjadi empat daerah anak dengan

1. $lebar\ daerah\ anak = lebar\ daerah / 2$
2. $tinggi\ daerah\ anak = tinggi\ daerah / 2$

Sehingga, luas daerah anak adalah $1/4$ luas daerah induk. Fungsi akan dipanggil kembali untuk setiap daerah anak sampai seluruh pixel dari frame gambar terisi.

Bagi asisten, dosen, dan mahasiswa, algoritma tersebut sudah tidak asing. Namun, dokumen ini juga ditujukan untuk pembaca yang awam mengenai fungsi rekursif. Untuk memudahkan visualisasi, penulis menyediakan *flowchart* sebagai berikut (perhitungan error diasumsikan sudah dilakukan).



Gambar 04 : *flowchart Divide and Conquer pada struktur data Quadtree untuk kompresi gambar.*
Dibuat dengan draw.io

Satu kotak di atas menandakan *scope* dari sebuah fungsi yang dapat dipanggil lagi dalam kotak yang sama, sehingga akan ada kotak dalam kotak jika dilakukan pemanggilan rekursif. Dalam struktur program yang dibuat, penulis mengimplementasikan algoritma *Divide and Conquer* dan pengisian warna dalam prosedur *buildRecursive* dan *reconstruct* dengan pseudocode sebagai berikut.

Metode buildRecursive

```

procedure buildRecursive(input image, x: integer, y: integer, width: integer, height: integer,
threshold: double, minSize: integer, method: ErrorMethod, currentDepth: integer)
→ QuadtreeNode
  
```

Deklarasi

```
avg : array of 3 elements untuk menyimpan warna dalam format [red, green, blue]  
currentArea, childArea : integer  
node : QuadtreeNode  
children : array[4] of QuadtreeNode
```

Algoritma

```
nodeCount ← nodeCount + 1 { Secara default menambah jumlah node setiap dipanggil }  
if currentDepth > maxDepth then  
    maxDepth ← currentDepth { Mengupdate maxDepth jika kedalaman saat ini maksimum }  
endif  
  
error ← hitung error sesuai metode yang dipilih, dengan memanggil fungsi computeError.  
  
{ Membuat node quadtree untuk daerah saat ini dan menetapkan nilai rata-rata warnanya }  
node ← buat quadTreeNode baru  
Set average merah, hijau, dan biru berdasarkan setiap elemen avg.  
  
{ Menghitung area saat ini dan area tiap anak (kuadran) }  
currentArea ← lebar dikali tinggi gambar  
childArea ← (width / 2) * (height / 2) { Luas saat ini dibagi 4 }  
  
{ Kasus Rekursif }  
if error > threshold and currentArea > minSize and childArea ≥ minSize then  
    node.setLeaf(false)  
    halfW ← width / 2  
    halfH ← height / 2  
    { Membagi gambar menjadi 4 dan membangun tree secara rekursif }  
    children[0] ← panggil buildRecursive untuk kuadran 1  
    children[1] ← panggil buildRecursive untuk kuadran 2  
    children[2] ← panggil buildRecursive untuk kuadran 3  
    children[3] ← panggil buildRecursive untuk kuadran 4  
    node.setChildren(children)  
endif  
→ node  
end procedure
```

Pseudocode di atas merupakan terjemahan secara literal dari *source code* dengan beberapa bagian program yang dijadikan kalimat manusia. Apabila diperhatikan dengan seksama, ada sedikit perbedaan pada implementasi program dengan rancangan awal (*flowchart*) yang menurut penulis masih wajar. Di sini, penentuan keputusan dilakukan secara kebalikan, rekursi tidak berada di dalam *else*, melainkan menjadi sebuah blok *if condition* dengan basisnya sebagai *default case*. Oleh karena itu, langkah-langkah

pendekatan yang dilakukan adalah sebagai berikut. Namun kedua implementasi artinya sama karena secara logika $p \equiv \neg(\neg p)$. Oleh karena itu, berikut merupakan urutan metode secara utuh.

1. Hitung rata-rata warna dan error menurut metode yang dipilih.
2. Jika nilai error melebihi threshold dan ukuran area dan ukuran anak (child) masih lebih besar dari minblock size yang telah ditentukan, gambar dibagi menjadi empat bagian secara rekursif.
3. Hasil dari pemrosesan masing-masing bagian dari gambar (subnode) kemudian digabungkan untuk membentuk Quadtree yang utuh, di mana setiap node memiliki referensi ke empat anaknya.

Metode reconstruct

```
procedure reconstruct(input output: BufferedImage, node: QuadtreeNode)
Deklarasi
    fillColor : Color

Algoritma
    { Jika node kosong, stop }
    if node is null then
        →
    endif

    { Jika node merupakan daun, gunakan nilai rata-rata untuk mengisi output }
    if node.isLeaf() then
        fillColor ← new Color(node.getAvgRed(), node.getAvgGreen(), node.getAvgBlue())
        for i ← node.getY() to (node.getY() + node.getHeight() - 1) do
            for j ← node.getX() to (node.getX() + node.getWidth() - 1) do
                Isi setiap warna untuk setiap koordinat pixel i dan j.
            end for
        end for
    else
        { Jika bukan daun, rekursif untuk setiap anak node }
        for each anak dari node do
            reconstruct(output, child)
        end for
    endif
end procedure
```

Langkah-langkah pendekatan yang dilakukan adalah sebagai berikut.

1. Gunakan quadtree yang telah dibentuk.
2. Jika node merupakan daun, nilai rata-rata warna dari node tersebut digunakan untuk mengisi piksel di area yang relevan.
3. Jika node bukan daun, fungsi dipanggil secara rekursif pada setiap anaknya sehingga keseluruhan gambar terrekonstruksi.

3.3 Implementasi dengan Java

3.3.1 Class QuadtreeNode

- Attribute

Nama	Tipe	Deskripsi
x, y	private int	Nilai yang menyimpan koordinat posisi node pada gambar (x,y)
width, height	private int	Nilai yang menyimpan lebar dan tinggi area yang direpresentasikan oleh node
avgRed, avgGreen, avgBlue	private int	Nilai rata-rata warna RGB untuk area yang direpresentasikan oleh node
isLeaf	private boolean	Menandakan apakah node merupakan leaf atau bukan
children	private QuadtreeNode[]	Array yang menyimpan 4 child node (pembagian quadtree)

```
public class QuadtreeNode {  
    private int x, y, width, height;  
    private int avgRed, avgGreen, avgBlue;  
    private boolean isLeaf;  
    private QuadtreeNode[] children;
```

- Method

Nama	Tipe	Parameter	Deskripsi
QuadtreeNode	public constructor	int x, int y, int width, int height	Membuat node baru dengan koordinat dan ukuran yang ditentukan
public QuadtreeNode(int x, int y, int width, int height) { this.x = x; this.y = y; this.width = width; this.height = height; this.isLeaf = true; this.children = new QuadtreeNode[4]; }			
getX	public int		Mengambil nilai

			koordinat x dari node
public int getX() { return x; }			
getY	public int		Mengambil nilai koordinat y dari node
public int getY() { return y; }			
getWidth	public int		Mengambil nilai lebar area node
public int getWidth() { return width; }			
getHeight	public int		Mengambil nilai tinggi area node
public int getHeight() { return height; }			
getAvgRed	public int		Mengambil nilai rata-rata komponen warna merah
public int getAvgRed() { return avgRed; }			
getAvgGreen	public int		Mengambil nilai rata-rata komponen warna hijau
public int getAvgGreen() { return avgGreen; }			
getAvgBlue	public int		Mengambil nilai rata-rata komponen warna biru

```

public int getAvgBlue() {
    return avgBlue;
}

```

setAvgRed	public void	int avgRed	Mengatur nilai rata-rata komponen warna merah
-----------	-------------	------------	---

```

public void setAvgRed(int avgRed) {
    this.avgRed = avgRed;
}

```

setAvgGreen	public void	int avgGreen	Mengatur nilai rata-rata komponen warna hijau
-------------	-------------	--------------	---

```

public void setAvgGreen(int avgGreen) {
    this.avgGreen = avgGreen;
}

```

setAvgBlue	public void	int avgBlue	Mengatur nilai rata-rata komponen warna biru
------------	-------------	-------------	--

```

public void setAvgBlue(int avgBlue) {
    this.avgBlue = avgBlue;
}

```

setColor	public void	int red, int green, int blue	Mengatur nilai warna RGB rata-rata node
----------	-------------	------------------------------	---

```

public void setColor(int red, int green, int blue) {
    this.avgRed = red;
    this.avgGreen = green;
    this.avgBlue = blue;
}

```

isLeaf	public boolean		Memeriksa apakah node merupakan leaf
--------	----------------	--	--------------------------------------

```

public boolean isLeaf() {
    return isLeaf;
}

```

setLeaf	public void	boolean isLeaf	Mengatur status leaf dari node
---------	-------------	----------------	--------------------------------

```

public void setLeaf(boolean isLeaf) {

```

	<pre>this.isLeaf = isLeaf;</pre>		
getChildren	<pre>public QuadtreeNode[]</pre>		Mengambil array child nodes
	<pre>public QuadtreeNode[] getChildren() { return children; }</pre>		
setChildren	<pre>public void</pre>	QuadtreeNode[] children	Mengatur array child nodes
	<pre>public void setChildren(QuadtreeNode[] children) { this.children = children; }</pre>		

3.3.2 Class Quadtree

- Attribute

Nama	Tipe	Deskripsi
root	private QuadtreeNode	Node akar dari struktur pohon quadtree
nodeCount	private int	Menyimpan jumlah total node dalam quadtree
maxDepth	private int	Menyimpan kedalaman maksimum dari quadtree
ErrorMethod	public enum	Enumerasi metode perhitungan error

```
public class Quadtree {
    private QuadtreeNode root;
    private int nodeCount;
    private int maxDepth;
    public enum ErrorMethod {
        VARIANCE, MAD, MAX_DIFF, ENTROPY
    }
}
```

- Method

Nama	Tipe	Parameter	Deskripsi

Quadtree	public constructor	-	Inisialisasi quadtree kosong dengan root null
	<pre>public Quadtree() { root = null; nodeCount = 0; maxDepth = 0; }</pre>		
getRoot	public QuadTreeNode		Mengambil node root dari quadtree
	<pre>public QuadTreeNode getRoot() { return root; }</pre>		
getNodeCount	public int	-	Mengambil jumlah total node
	<pre>public int getNodeCount() { return nodeCount; }</pre>		
getMaxDepth	public int	-	Mengambil kedalaman maksimum tree
	<pre>public int getMaxDepth() { return maxDepth; }</pre>		
build	public void	BufferedImage image, int x, int y, int width, int height, double threshold, int minSize, ErrorMethod method, ImageProcessor processor, boolean generateGIF)	Membangun quadtree dari gambar input dan menghasilkan frame-frame GIF jika diminta
	<pre>public void build(BufferedImage image, int x, int y, int width, int height, double threshold, int minSize, ErrorMethod method, ImageProcessor processor, boolean generateGIF) { nodeCount = 0; maxDepth = 0; root = buildRecursive(image, x, y, width, height, threshold,</pre>		

```

minSize, method, 1);

    if (generateGIF) {
        for (int d = 1; d <= maxDepth; d++) {
            BufferedImage stepImage = new
BufferedImage(image.getWidth(), image.getHeight(), image.getType());
            processor.setOutputImage(stepImage);
            reconstructUntilDepth(stepImage, root, d, 1);
            String filename =
String.format("../test/frames/depth_%02d.png", d);
            boolean saved = processor.saveImageFrame(filename,
stepImage);
            if (!saved) {
                System.out.println("Gagal menyimpan frame untuk
depth " + d);
            }
        }
    }
}

```

reconstructUntilDepth	public void	BufferedImage output, QuadtreeNode node, int maxAllowedDepth, int currentDepth	Merekonstruksi gambar sampai kedalaman tertentu
-----------------------	-------------	---	---

```

public void reconstructUntilDepth(BufferedImage output, QuadtreeNode
node, int maxAllowedDepth, int currentDepth) {
    if (node == null) return;
    if (node.isLeaf() || currentDepth >= maxAllowedDepth) {
        Color fill = new Color(node.getAvgRed(), node.getAvgGreen(),
node.getAvgBlue());
        for (int i = node.getY(); i < node.getY() +
node.getHeight(); i++) {
            for (int j = node.getX(); j < node.getX() +
node.getWidth(); j++) {
                output.setRGB(j, i, fill.getRGB());
            }
        }
    } else {
        for (QuadtreeNode child : node.getChildren()) {
            reconstructUntilDepth(output, child, maxAllowedDepth,

```

```
currentDepth + 1);  
    }  
}  
}
```

buildRecursive	private QuadtreeNode	BufferedImage image, int x, int y, int width, int height, double threshold, int minSize, ErrorMethod method, int currentDepth	Membangun quadtree secara rekursif (membagi node menjadi 4 bagian)
----------------	----------------------	--	---

```

    }
    return node;
}

computeError    private double      BufferedImage image,
                int x, int y, int width,
                int height, ErrorMethod
                method, double[] avg   Menghitung error area
                                         image menggunakan
                                         metode yang dipilih

```

```

private double computeError(BufferedImage image, int x, int y, int
width, int height,
                           ErrorMethod method, double[] avg) {
    int count = width * height;
    double sumR = 0, sumG = 0, sumB = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            Color color = new Color(image.getRGB(j, i));
            sumR += color.getRed();
            sumG += color.getGreen();
            sumB += color.getBlue();
        }
    }
    avg[0] = sumR / count;
    avg[1] = sumG / count;
    avg[2] = sumB / count;
    double error = 0.0;
    switch (method) {
        case VARIANCE:
            double varR = 0, varG = 0, varB = 0;
            for (int i = y; i < y + height; i++) {
                for (int j = x; j < x + width; j++) {
                    Color color = new Color(image.getRGB(j, i));
                    varR += Math.pow(color.getRed() - avg[0], 2);
                    varG += Math.pow(color.getGreen() - avg[1], 2);
                    varB += Math.pow(color.getBlue() - avg[2], 2);
                }
            }
            error = (varR + varG + varB) / (3 * count);
            break;
    }
}

```

```

    case MAD:
        double madR = 0, madG = 0, madB = 0;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                Color color = new Color(image.getRGB(j, i));
                madR += Math.abs(color.getRed() - avg[0]);
                madG += Math.abs(color.getGreen() - avg[1]);
                madB += Math.abs(color.getBlue() - avg[2]);
            }
        }
        error = (madR + madG + madB) / (3 * count);
        break;
    case MAX_DIFF:
        int minR = 255, minG = 255, minB = 255;
        int maxR = 0, maxG = 0, maxB = 0;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                Color color = new Color(image.getRGB(j, i));
                int r = color.getRed(), g = color.getGreen(), b
= color.getBlue();
                if (r < minR) minR = r;
                if (g < minG) minG = g;
                if (b < minB) minB = b;
                if (r > maxR) maxR = r;
                if (g > maxG) maxG = g;
                if (b > maxB) maxB = b;
            }
        }
        error = ((maxR - minR) + (maxG - minG) + (maxB - minB))
/ 3.0;
        break;
    case ENTROPY:
        double entropySum = 0.0;
        for (int c = 0; c < 3; c++) {
            int[] hist = new int[256];
            for (int i = y; i < y + height; i++) {
                for (int j = x; j < x + width; j++) {
                    Color color = new Color(image.getRGB(j, i));
                    int value = (c == 0) ? color.getRed() : (c

```

```

== 1 ? color.getGreen() : color.getBlue());
                hist[value]++;
}
}
double entropy = 0.0;
for (int k = 0; k < 256; k++) {
    if (hist[k] > 0) {
        double p = (double) hist[k] / count;
        entropy -= p * (Math.log(p) / Math.log(2));
    }
}
entropySum += entropy;
}
error = entropySum / 3.0;
break;
}
return error;
}

```

reconstruct	public void	BufferedImage output, QuadtreeNode node	Merekonstruksi gambar lengkap dari quadtree
-------------	-------------	--	---

```

public void reconstruct(BufferedImage output, QuadtreeNode node) {
    if (node == null) return;
    if (node.isLeaf()) {
        Color fill = new Color(node.getAvgRed(), node.getAvgGreen(),
node.getAvgBlue());
        for (int i = node.getY(); i < node.getY() +
node.getHeight(); i++) {
            for (int j = node.getX(); j < node.getX() +
node.getWidth(); j++) {
                output.setRGB(j, i, fill.getRGB());
            }
        }
    } else {
        for (QuadtreeNode child : node.getChildren()) {
            reconstruct(output, child);
        }
    }
}

```

3.3.3 Class ImageProcessor

- Attribute

Nama	Tipe	Deskripsi
inputImage	private BufferedImage	Menyimpan gambar input yang akan diproses
outputImage	private BufferedImage	Menyimpan gambar hasil pemrosesan
<pre>public class ImageProcessor { private BufferedImage inputImage; private BufferedImage outputImage;</pre>		

- Method

Nama	Tipe	Parameter	Deskripsi
ImageProcessor	public constructor		Membuat instance baru dengan nilai input dan output null
<pre>public ImageProcessor() { inputImage = null; outputImage = null; }</pre>			
getInputImage	public BufferedImage		Mengambil gambar input yang tersimpan
<pre>public BufferedImage getInputImage() { return inputImage; }</pre>			
getOutputImage	public BufferedImage		Mengambil gambar input yang tersimpan
<pre>public BufferedImage getOutputImage() { return outputImage; }</pre>			
setInputImage	public void		Mengatur gambar input yang akan diproses
<pre>public void setInputImage(BufferedImage inputImage) {</pre>			

```
        this.inputImage = inputImage;
    }
```

setOutputImage	public void		Mengatur gambar output hasil pemrosesan
----------------	-------------	--	---

```
public void setOutputImage(BufferedImage outputImage) {
    this.outputImage = outputImage;
}
```

loadImage	public boolean	String path	Memuat gambar dari file ke inputImage, bernilai True jika berhasil dan False jika gagal
-----------	----------------	-------------	---

```
public boolean loadImage(String path) {
    try {
        inputImage = ImageIO.read(new File(path));
        return inputImage != null;
    } catch (Exception e) {
        return false;
    }
}
```

saveImage	public boolean	String path	Menyimpan outputImage ke file, bernilai True jika berhasil dan False jika gagal
-----------	----------------	-------------	---

```
public boolean saveImage(String path) {
    try {
        String format = path.substring(path.lastIndexOf('.') +
1).toLowerCase();
        if (!format.matches("png|jpg|jpeg")) {
            format = "png";
        }
        return ImageIO.write(outputImage, format, new File(path));
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

saveImageFrame	public boolean	String path, BufferedImage image	Menyimpan frame gambar tertentu ke file PNG, bernilai True jika berhasil dan False jika gagal
<pre>public boolean saveImageFrame(String path, BufferedImage image) { try { return ImageIO.write(image, "png", new File(path)); } catch (Exception e) { return false; } }</pre>			
getFileSize	public long	String path	Mendapatkan ukuran file dalam bytes
<pre>public long getFileSize(String path) { File file = new File(path); return file.length(); }</pre>			

3.4 Implementasi Bonus (GIF Generator)

Secara umum, pembuatan GIF dilakukan secara tradisional, yaitu menyimpan frame gambar total setiap iterasi rekursi (pada *Quadtree.java*) pada sebuah folder sementara, penulis menggunakan folder test/frames. Karena frame disimpan setiap terjadi rekursi yang menimbulkan *depth* baru, artinya gambar disimpan di setiap *depth*. Sehingga *jumlah frame = kedalaman tree*.

3.4.1 Class GifGenerator

- Method

Nama	Tipe	Parameter	Deskripsi
generateGIF	public static void	String framesDir, String outputGifPath, int delayMS	Membaca file-file gambar frame dari direktori test/frames, mengurutkannya berdasarkan nama, dan menyatukan gambar-gambar tersebut menjadi sebuah file .gif

```

public static void generateGif(String framesDir, String outputGifPath,
int delayMS) {
    try {
        File dir = new File(framesDir);
        File[] imageFiles = dir.listFiles(new FilenameFilter() {
            public boolean accept(File dir, String name) {
                return name.startsWith("depth_") &&
name.endsWith(".png");
            }
        });
        if (imageFiles == null || imageFiles.length == 0) {
            System.out.println("Tidak ada frame PNG ditemukan di " +
framesDir);
            return;
        }
        Arrays.sort(imageFiles, Comparator.comparing(File::getName));
        BufferedImage firstImage = ImageIO.read(imageFiles[0]);
        ImageOutputStream output = new FileImageOutputStream(new
File(outputGifPath));
        GifSequence writer = new GifSequence(output,
firstImage.getType(), delayMS);
        writer.writeToSequence(firstImage);
        for (int i = 1; i < imageFiles.length; i++) {
            BufferedImage nextImage = ImageIO.read(imageFiles[i]);
            writer.writeToSequence(nextImage);
        }
        writer.close();
        output.close();
    } catch (Exception e) {
        return;
    }
}

```

3.4.2 Class GifSequence

- Attribute

Nama	Tipe	Deskripsi
gifWriter	protected ImageWriter	Objek sementara untuk menyimpan frame gambar.
imageWriteParam	protected ImageWriteParam	Atribut penampung parameter untuk menulis gambar (akan dipanggil dengan gifWriter.getDefaultWriteParam()).
imageMetaData	protected IIOMetadata	Menampung metadata untuk GIF sendiri, misalnya delay, loop, ataupun app extension (disini loop sudah ditetapkan infinite {0x01, 0x00, 0x00}).

```
public class GifSequence {
    protected ImageWriter gifWriter;
    protected ImageWriteParam imageWriteParam;
    protected IIOMetadata imageMetaData;
```

- Method

Nama	Tipe	Parameter	Deskripsi
GifSequence	public	ImageOutputStream outputStream, int imageType, int delay	Konstruktor kelas GifSequence termasuk inisialisasi gifWriter, pengaturan delay, dan infinite loop.

```
public GifSequence(ImageOutputStream outputStream, int imageType, int
delay) throws IOException {
    gifWriter = ImageIO.getImageWritersBySuffix("gif").next();
    imageWriteParam = gifWriter.getDefaultWriteParam();
    ImageTypeSpecifier imageTypeSpecifier =
    ImageTypeSpecifier.createFromBufferedImageType(imageType);
    imageMetaData =
```

```

gifWriter.getDefaultImageMetadata(imageTypeSpecifier, imageWriteParam);

    String metadataFormatName =
imageMetaData.getNativeMetadataFormatName();

    IIOMetadataNode root = (IIOMetadataNode)
imageMetaData.getAsTree(metadataFormatName);

    IIOMetadataNode graphicsControlExtensionNode = getNode(root,
"GraphicControlExtension");

        graphicsControlExtensionNode.setAttribute("disposalMethod",
"none");

        graphicsControlExtensionNode.setAttribute("userInputFlag",
"FALSE");

        graphicsControlExtensionNode.setAttribute("transparentColorFlag",
"FALSE");

        graphicsControlExtensionNode.setAttribute("delayTime",
Integer.toString(delay / 10));

graphicsControlExtensionNode.setAttribute("transparentColorIndex", "0");

    IIOMetadataNode appExtensionsNode = getNode(root,
"ApplicationExtensions");

    IIOMetadataNode appExtensionNode = new
IIOMetadataNode("ApplicationExtension");

        appExtensionNode.setAttribute("applicationID", "NETSCAPE");
        appExtensionNode.setAttribute("authenticationCode", "2.0");
        appExtensionNode.setUserObject(new byte[]{0x1, 0x00, 0x00});
        appExtensionsNode.appendChild(appExtensionNode);
        root.appendChild(appExtensionsNode);
        imageMetaData.setFromTree(metadataFormatName, root);

    gifWriter.setOutput(outputStream);

    gifWriter.prepareWriteSequence(null);

}

```

writeToSequence	public void	RenderedImage img	Menulis satu frame gambar ke dalam <i>sequence</i> GIF yang sedang dibangun.
-----------------	-------------	----------------------	--

```

public void writeToSequence(RenderedImage img) throws IOException {
    gifWriter.writeToSequence(new IIIOImage(img, null, imageMetaData),

```

```

imageWriteParam);
}

```

close	public void	-	Menghentikan <i>sequence</i> penulisan gambar atau menyelesaikan pembuatan GIF.
-------	-------------	---	---

```

public void close() throws IOException {
    gifWriter.endWriteSequence();
}

```

getNode	private static IIOMetadataNode rootNode, String nodeName	IIOMetadataNode rootNode, String nodeName	Mencari dan mengembalikan node metadata dari nama nodenya. Jika ternyata node tersebut belum ada, buat dan tambah node yang dimaksud ke root.
---------	--	---	---

```

private static IIOMetadataNode getNode(IIOMetadataNode rootNode, String
nodeName) {
    for (int i = 0; i < rootNode.getLength(); i++) {
        if
(rootNode.item(i).getNodeName().equalsIgnoreCase(nodeName)) {
            return (IIOMetadataNode) rootNode.item(i);
        }
    }
    IIOMetadataNode node = new IIOMetadataNode(nodeName);
    rootNode.appendChild(node);
    return node;
}

```

Tabel 02 : Tabel penjelasan algoritma.

Metode untuk menyimpan gambar secara langsung ke folder test/frames tidak dideklarasikan secara eksplisit, melainkan pada metode *build* di Quadtree.java.

4 Hasil dan Pembahasan

4.1 Kompleksitas Algoritma

Untuk menentukan kompleksitas waktu pada algoritma yang telah dibuat, khususnya pada bagian *divide and conquer*, dapat dianalisis dari metode buildRecursive. Algoritma ini akan membagi gambar menjadi empat bagian secara rekursif dan setiap pembagian dilakukan apabila nilai error melebihi threshold dan ukuran blok setelah dibagi empat tidak lebih kecil dari size blok minimum yang telah ditentukan.

Jumlah maksimum node dalam struktur pohon quadtree (pembagian penuh hingga tiap blok $n \times n$) dapat dituliskan sebagai berikut.

$$T(n) = 1 + 4 + 4^2 + \dots + 4^{\log_2 n} = \sum_{k=0}^{\log_2 n} 4^k = \frac{4^{\log_2 n + 1} - 1}{4 - 1}$$

$$\text{Karena } 4^{\log_2 n} = n^2, \text{ maka } T(n) = O(n^2)$$

Dan untuk setiap node, dilakukan perhitungan error dalam $O(k^2)$ waktu, dimana k adalah ukuran sisi blok saat itu. Ukuran blok pada depth ke- k dapat dihitung sebagai berikut.

- Pada depth ke-0, saat gambar masih belum dibagi, blok akan berukuran $n \times n$
- Pada depth ke-1, saat gambar dibagi menjadi empat bagian, blok akan berukuran $\frac{n}{2} \times \frac{n}{2}$
- Pada depth ke-2, setiap blok akan dibagi menjadi empat bagian, sehingga akan memiliki total 16 blok dan masing-masing blok akan berukuran $\frac{n}{4} \times \frac{n}{4}$
- Dan seterusnya.

Sehingga secara umum, pada depth ke- k , ukuran sisi dari setiap blok adalah $\frac{n}{2^k}$ dan luas per blok akan berukuran $(\frac{n}{2^k})^2$. Karena ukuran blok menurun secara eksponensial, maka total kompleksitas waktu dengan mempertimbangkan pembagian dan perhitungan error untuk setiap blok dapat dituliskan sebagai berikut.

$$T(n) = \sum_{k=0}^{\log_2 n} 4^k \cdot (\frac{n}{2^k})^2 = \sum_{k=0}^{\log_2 n} 4^k \cdot \frac{n^2}{4^k} = \sum_{k=0}^{\log_2 n} n^2 = (\log_2 n + 1) \cdot n^2$$

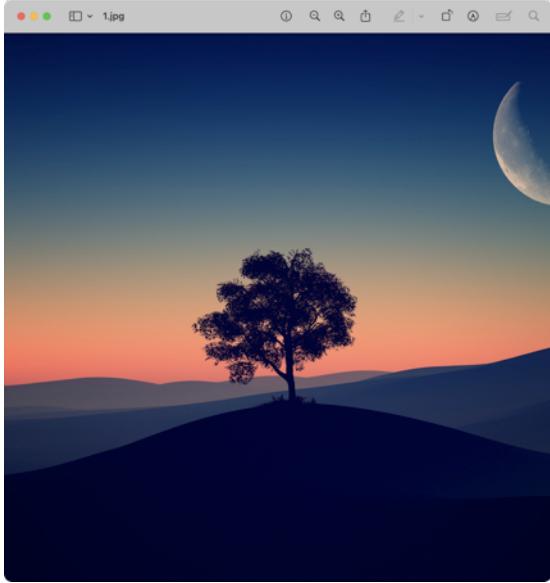
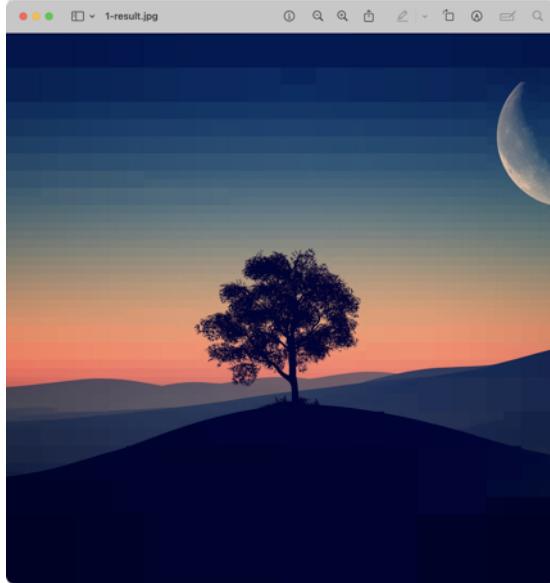
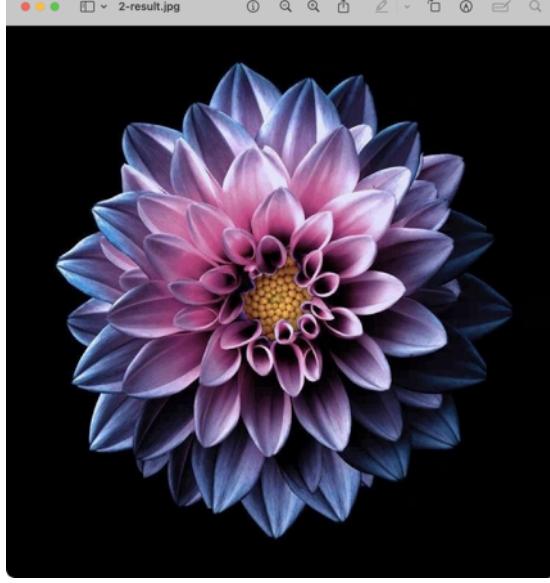
Berdasarkan perhitungan di atas, dapat disimpulkan bahwa kompleksitas waktu untuk algoritma yang telah diimplementasikan bernilai:

$$T(n) = O(n^2 \log n)$$

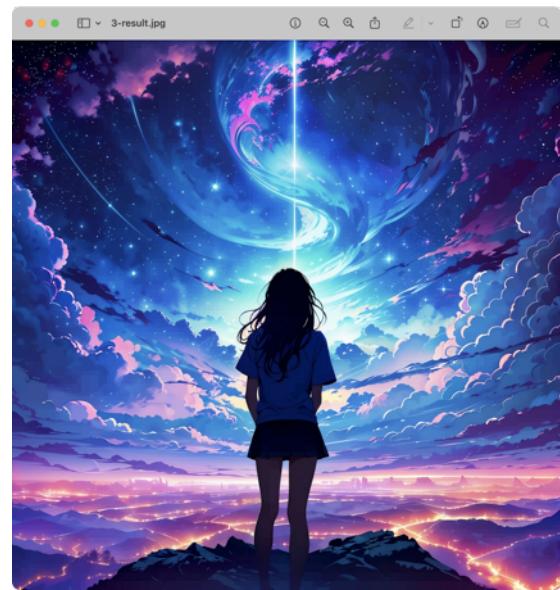
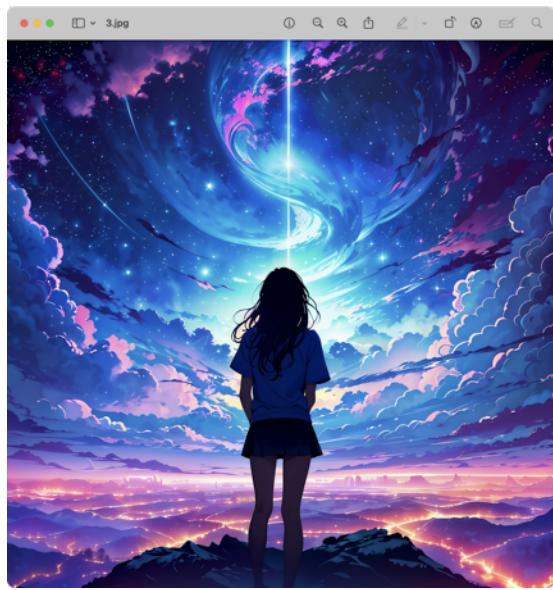
4.2 Pengujian Algoritma

4.2.1 Variance

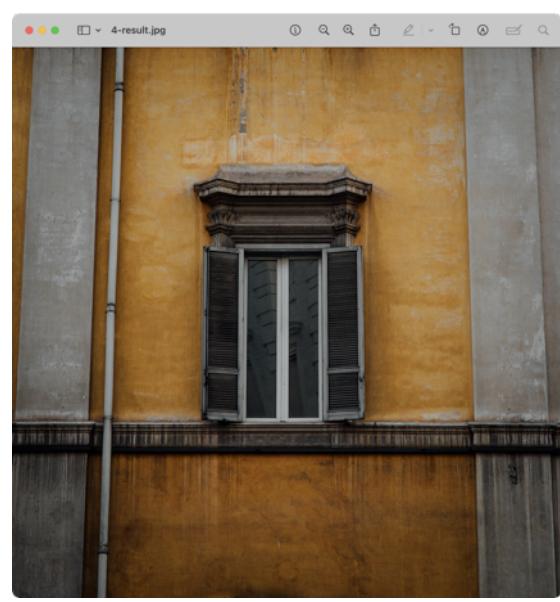
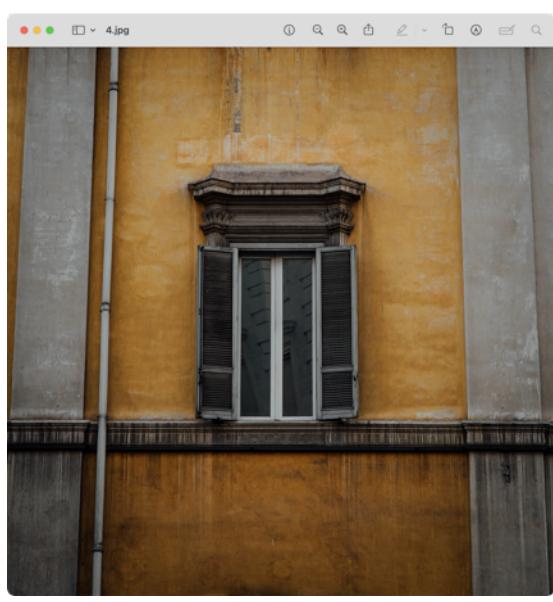
Nilai *threshold* dibuat menjadi 15 dengan *minimum block size* bernilai 1. Setelan terus dipilih karena menghasilkan hasil kompresi yang rata-rata lebih dari 50% untuk gambar 4K dengan kualitas yang cenderung masih tajam. Selain itu, nilai *threshold* dan *minimum block size* ini kongruen dengan pengaturan untuk metode lain, sehingga menghasilkan perbandingan yang lebih *apple-to-apple*.

No.	Gambar Awal	Gambar Hasil
1.		
2.		

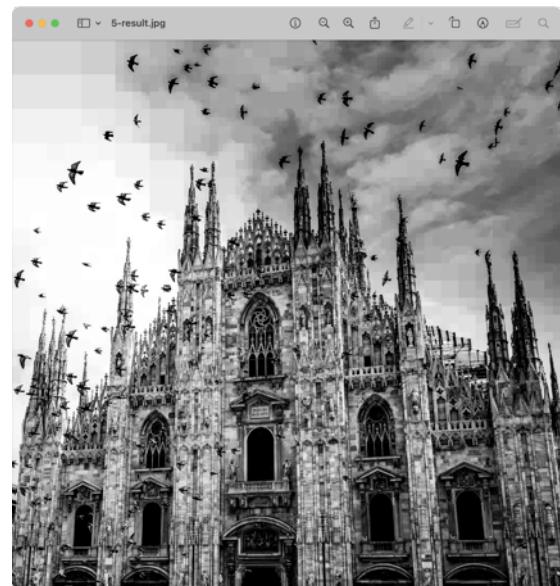
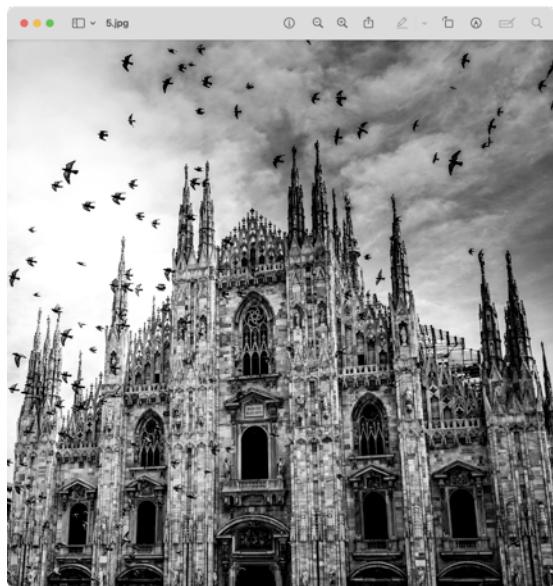
3.



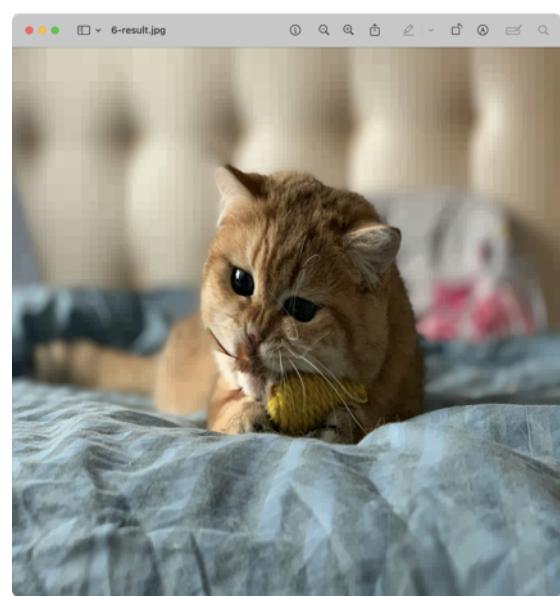
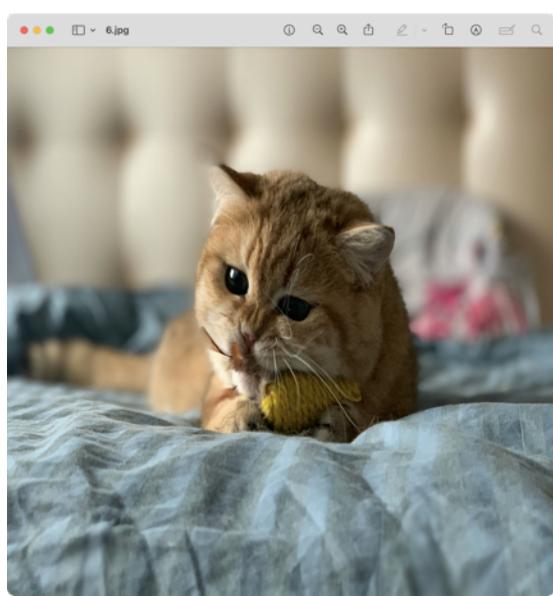
4.

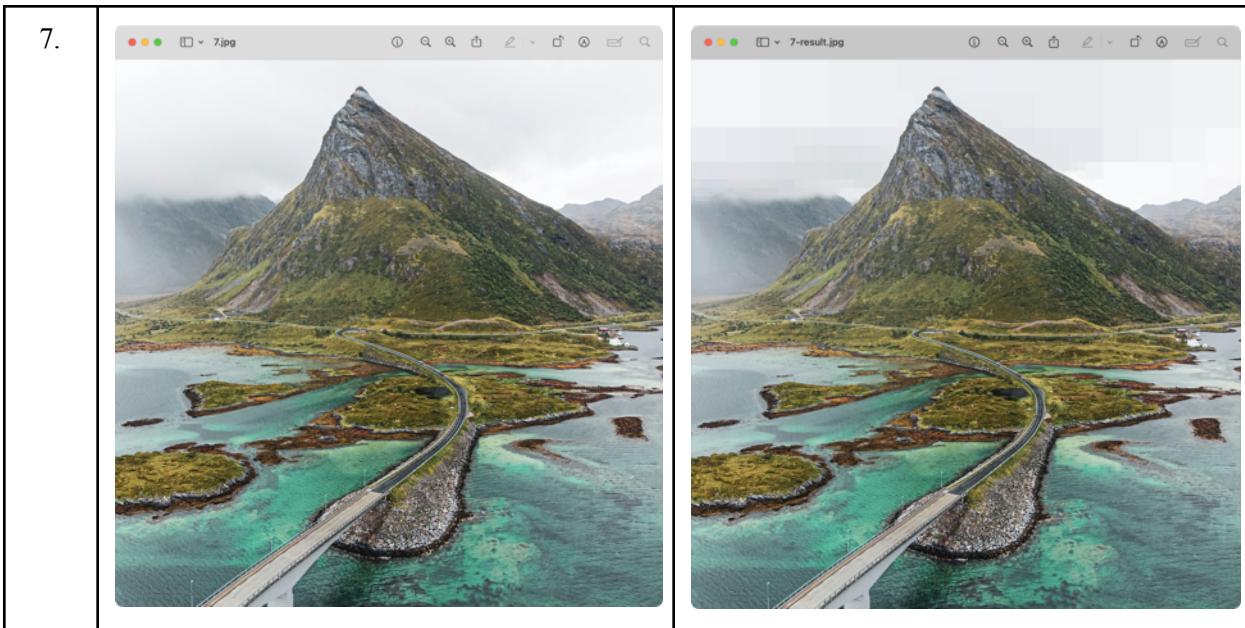


5.



6.





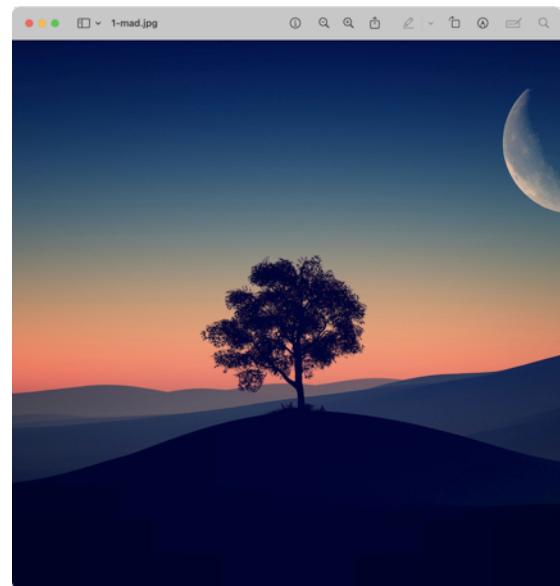
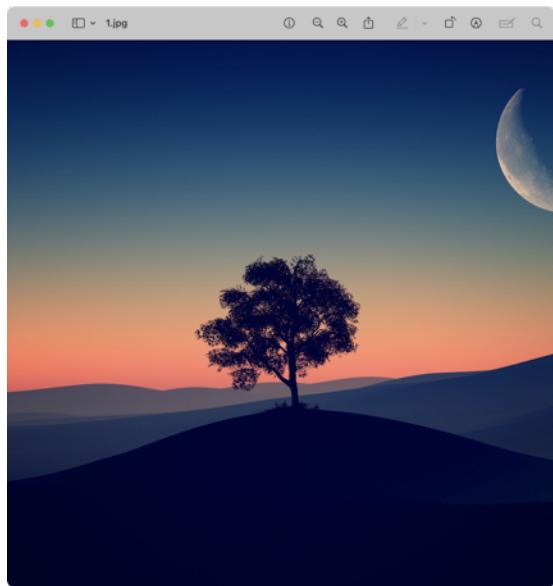
No.	Ukuran Awal (KB)	Ukuran Akhir (KB)	Kompresi (%)
1	942	194	79,40
2	139	64	53,99
3	1277	509	60,18
4	2669	1539	42,35
5	2057	1903	7,48
6	1262	590	53,25
7	1955	1259	35,57

4.2.2 Mean Absolute Deviation (MAD)

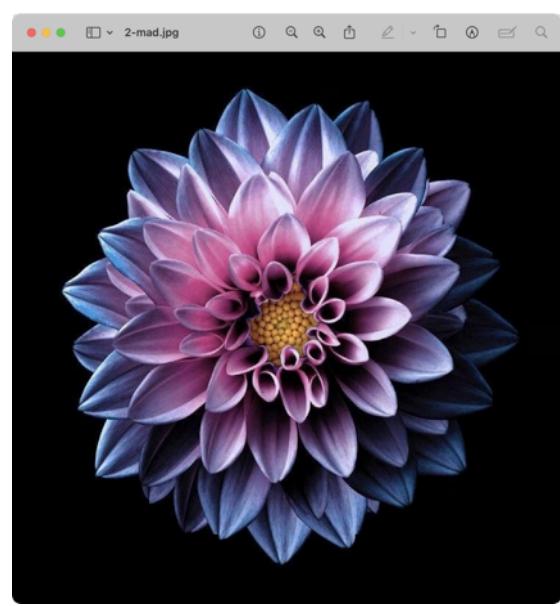
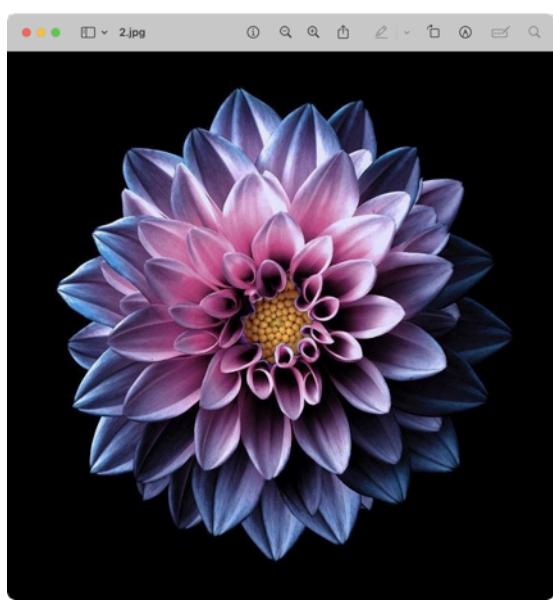
Nilai *threshold* dan *minimum block size* di set menjadi 1 untuk mendapatkan hasil terbaik atau kompresi paling tidak ekstrem. Nilai-nilai tersebut kongruen dengan pengaturan metode Variance dan menghasilkan hasil kompresi yang cenderung sama.

No.	Gambar Awal	Gambar Hasil

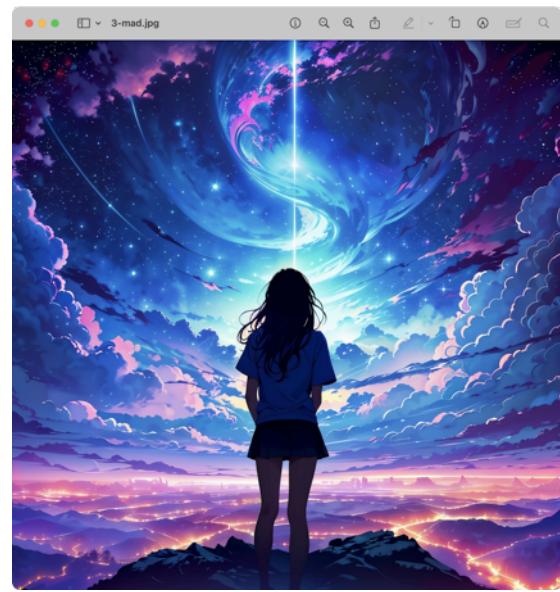
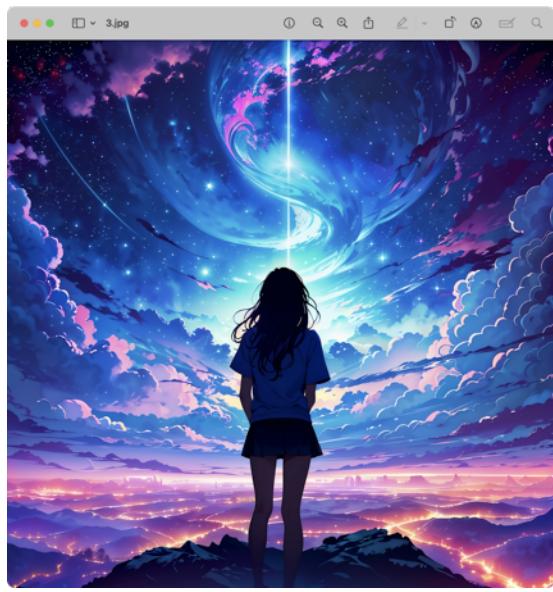
1.



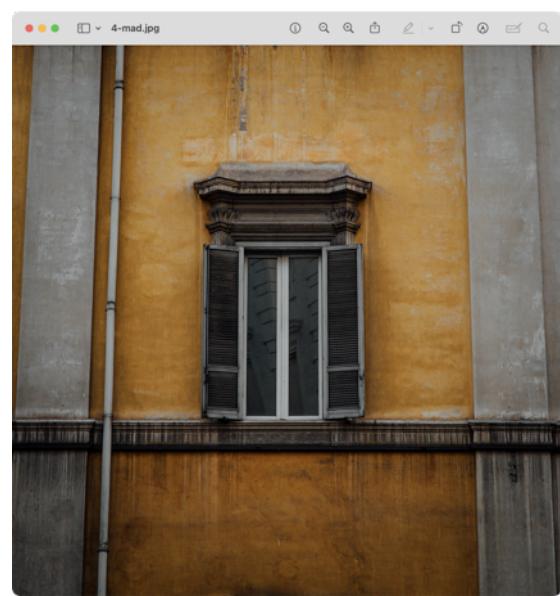
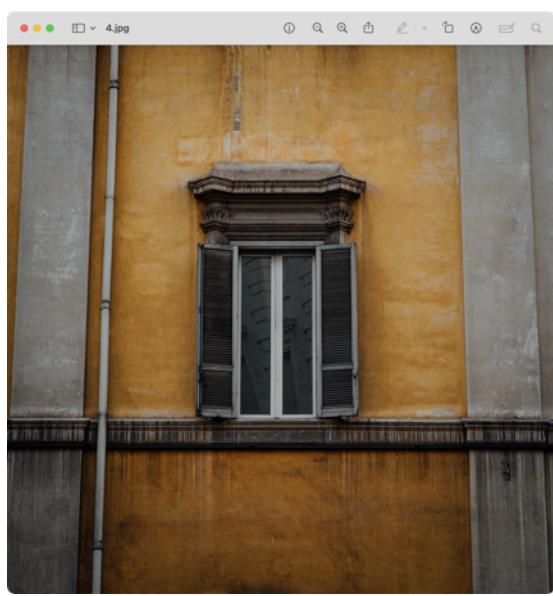
2.



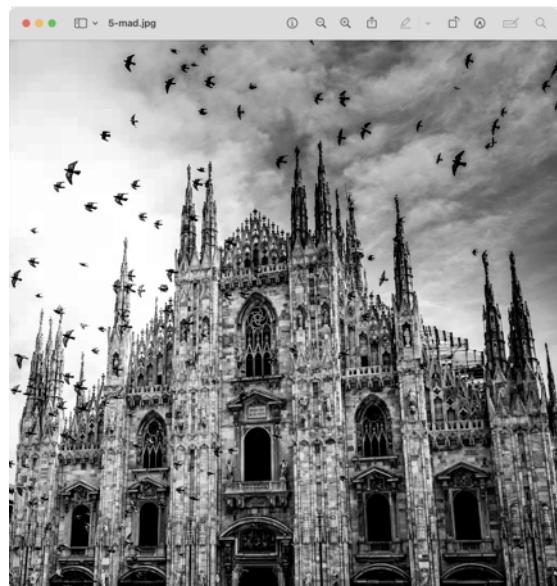
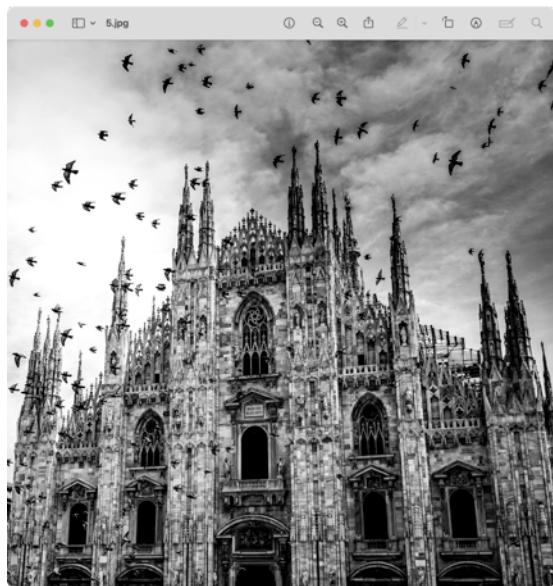
3.



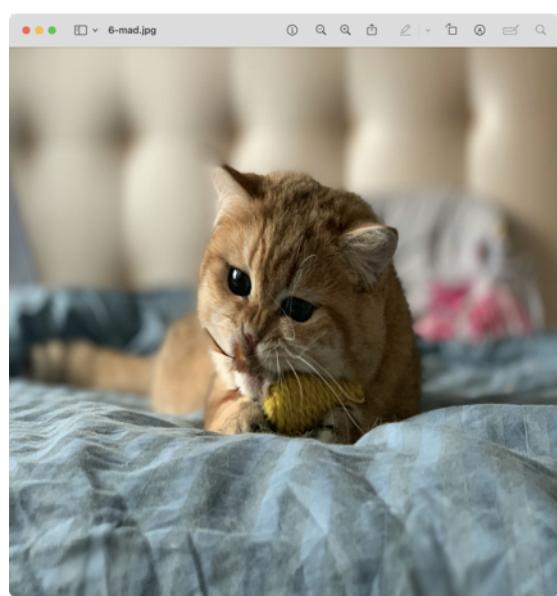
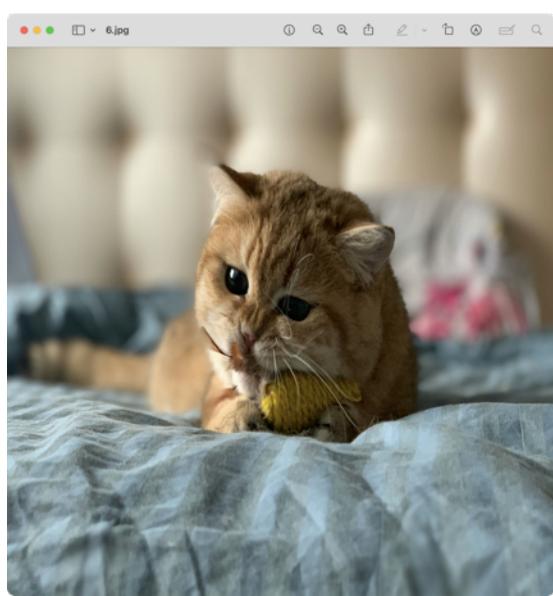
4.

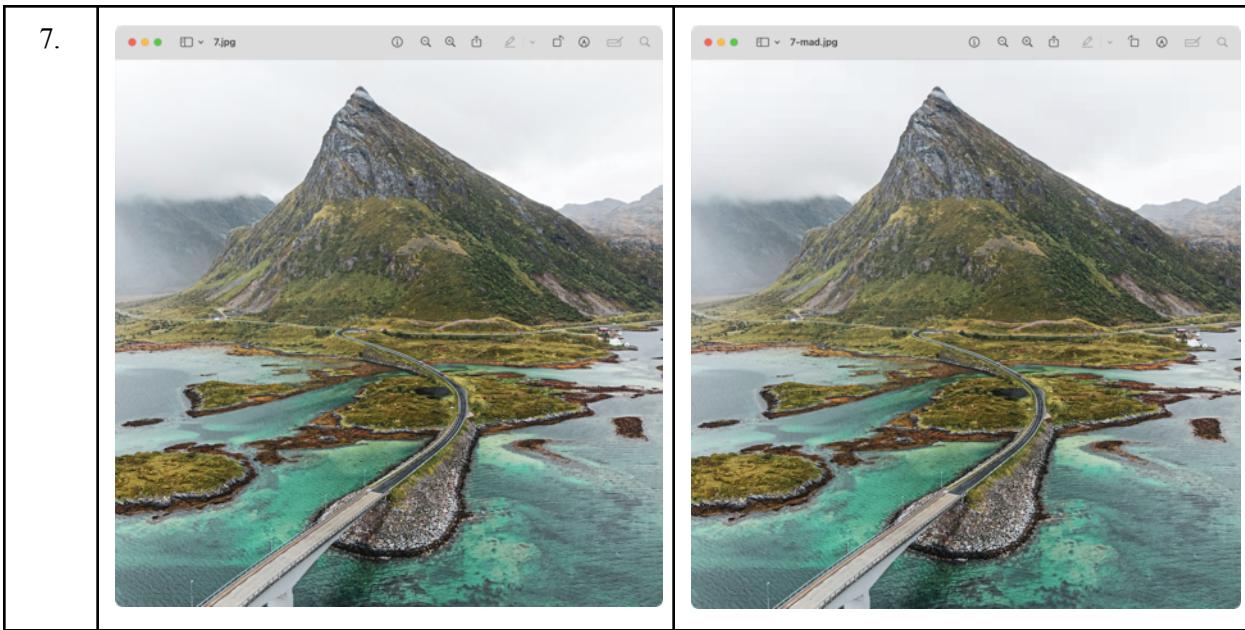


5.



6.





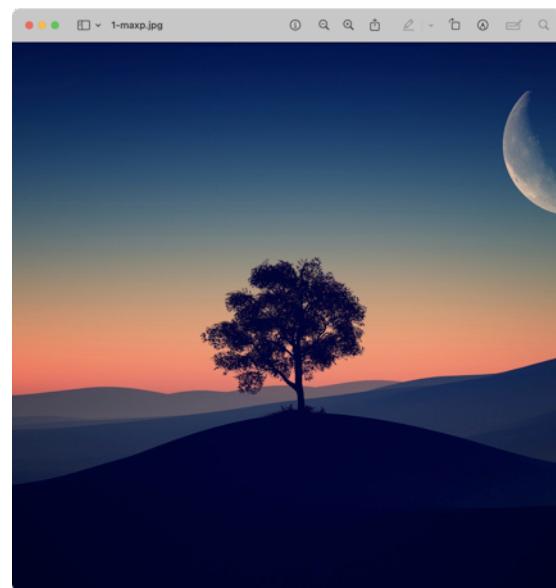
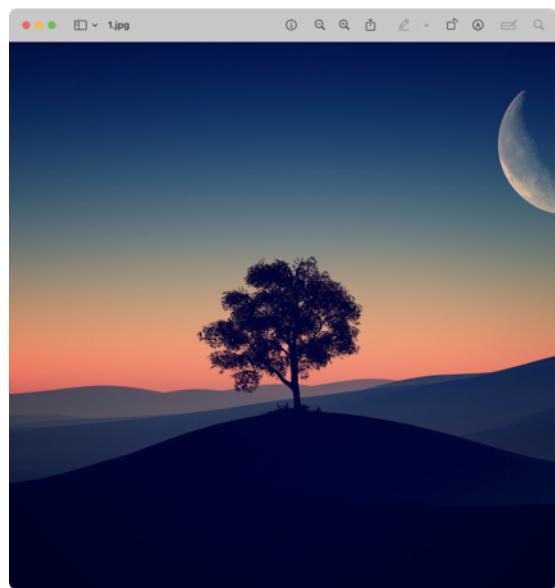
No.	Ukuran Awal (KB)	Ukuran Akhir (KB)	Kompresi (%)
1	942	245	73,91
2	139	65	53,33
3	1277	501	60,78
4	2669	1551	41,90
5	2057	2043	0,65
6	1263	664	47,44
7	1955	1270	35,04

4.2.3 Max Pixel Difference

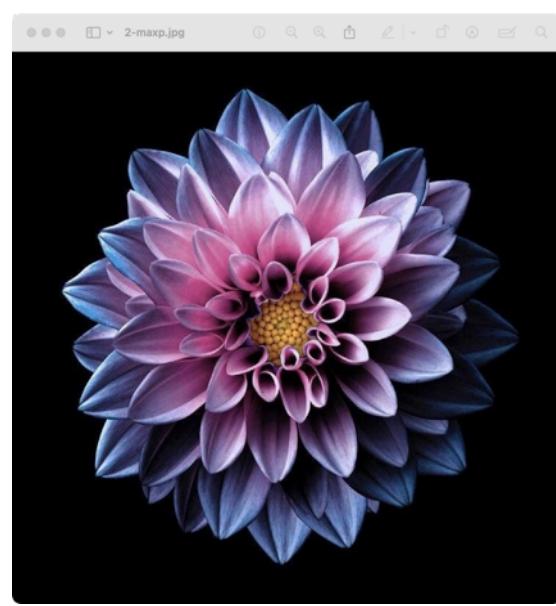
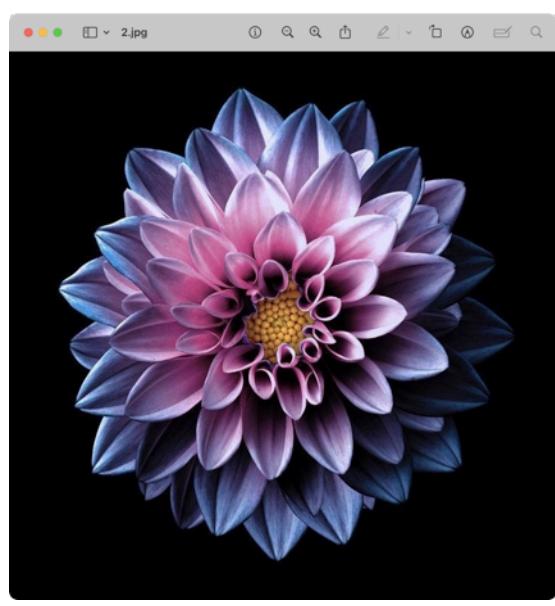
Nilai *threshold* dan *minimum block size* di set menjadi 1 untuk mendapatkan hasil terbaik atau kompresi paling tidak ekstrem. Nilai-nilai tersebut kongruen dengan pengaturan metode Variance dan menghasilkan hasil kompresi yang cenderung sama.

No.	Gambar Awal	Gambar Hasil
-----	-------------	--------------

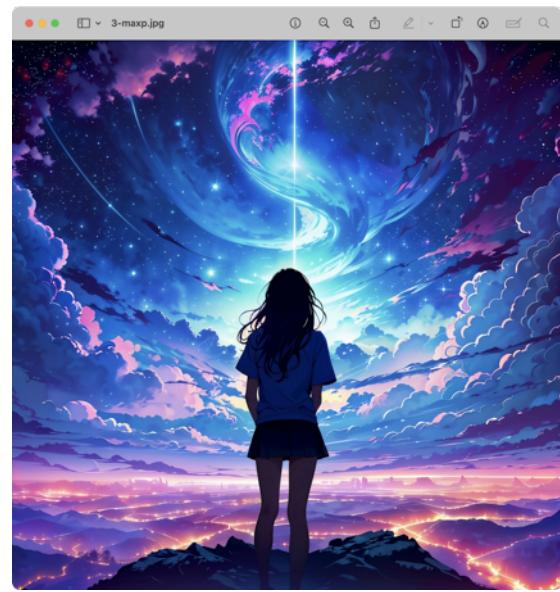
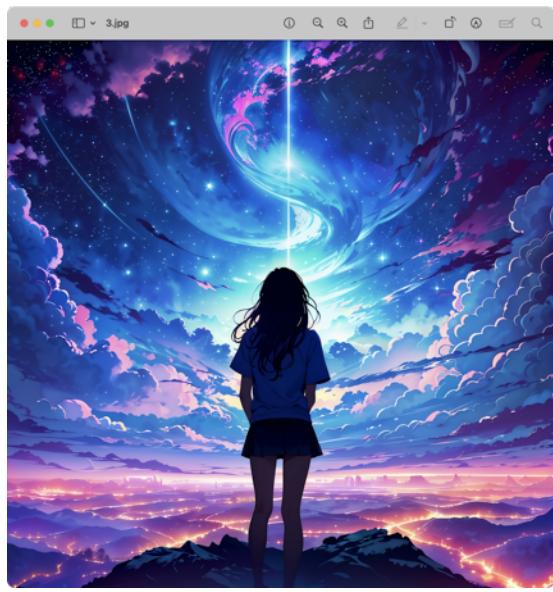
1.



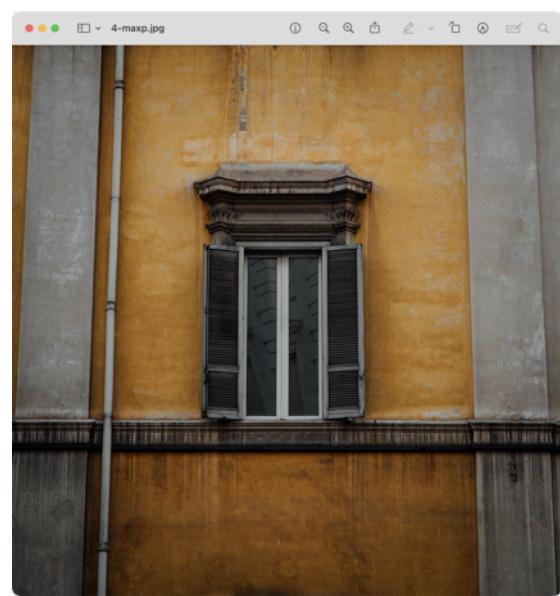
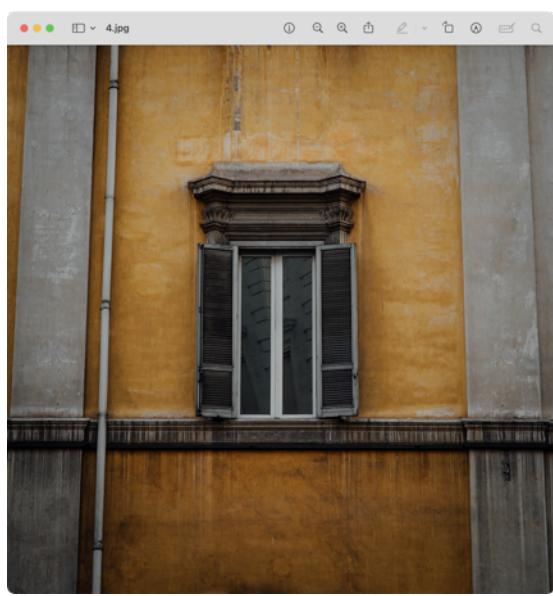
2.



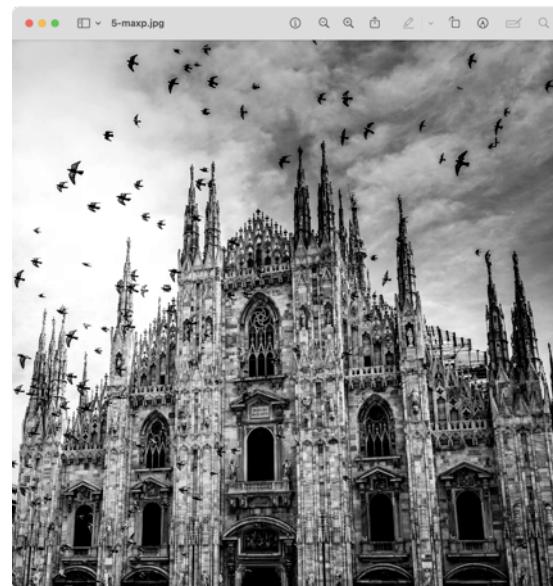
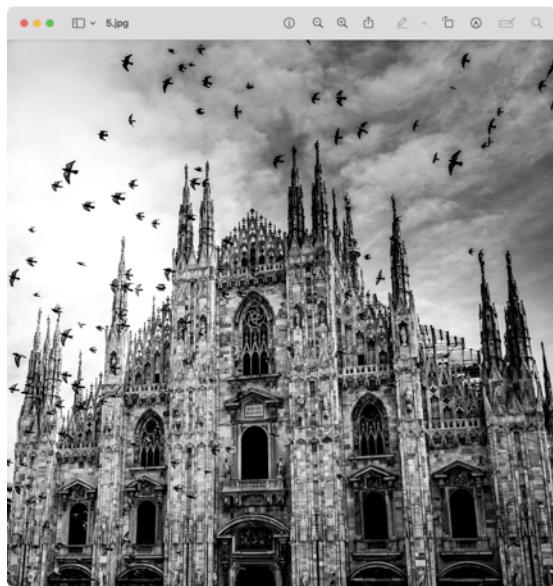
3.



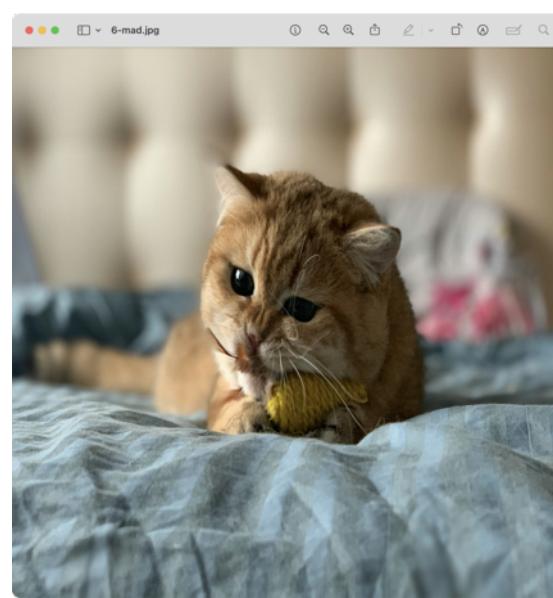
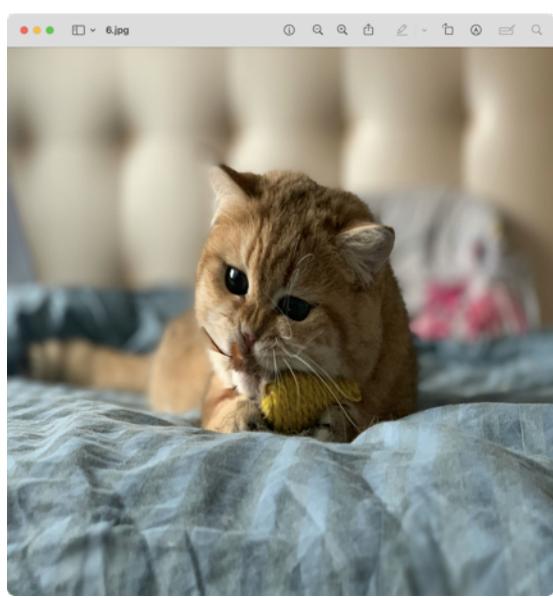
4.

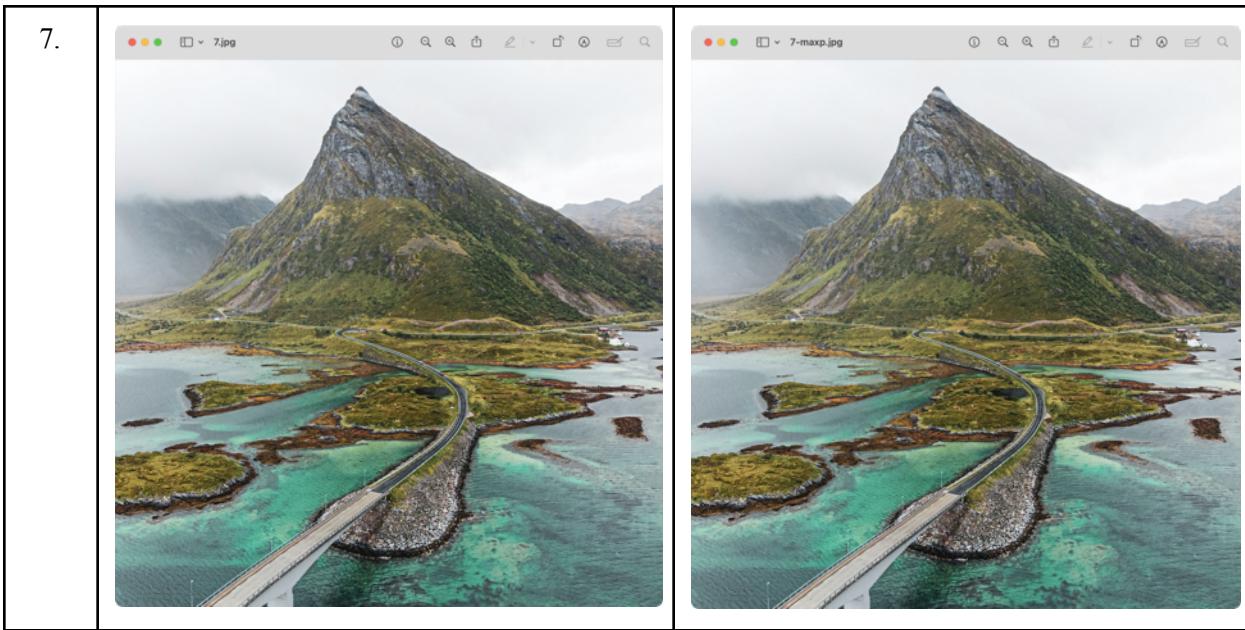


5.



6.





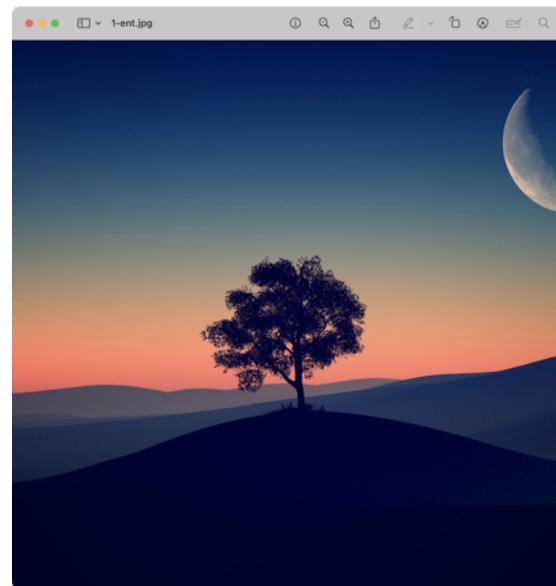
No.	Ukuran Awal (KB)	Ukuran Akhir (KB)	Kompresi (%)
1	942	264	71,93
2	139	65	53,31
3	1277	503	60,58
4	2669	1551	41,90
5	2057	2089	-1,56
6	1263	683	45,87
7	1955	1274	34,84

4.2.4 Entropy

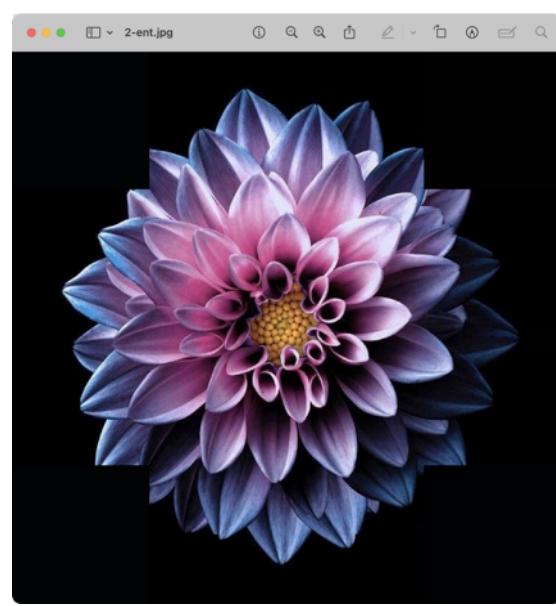
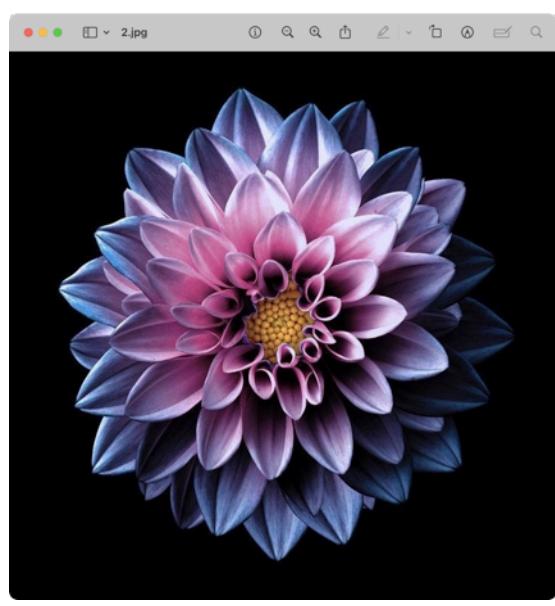
Nilai *threshold* dan *minimum block size* di set menjadi 1 untuk mendapatkan hasil terbaik atau kompresi paling tidak ekstrem. Nilai-nilai tersebut kongruen dengan pengaturan metode Variance dan menghasilkan hasil kompresi yang cenderung sama.

No.	Gambar Awal	Gambar Hasil
-----	-------------	--------------

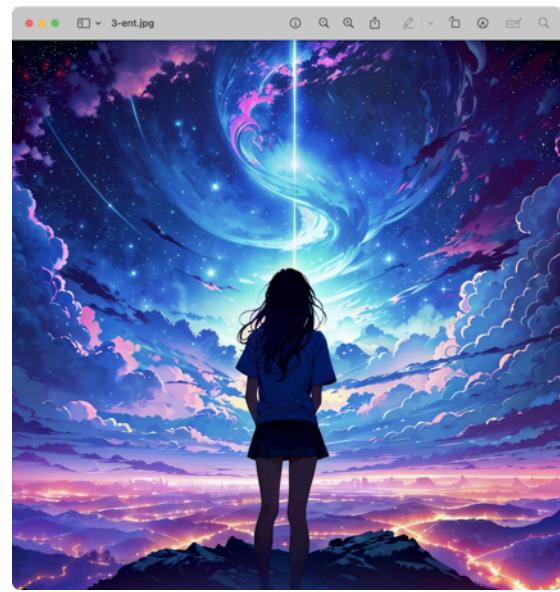
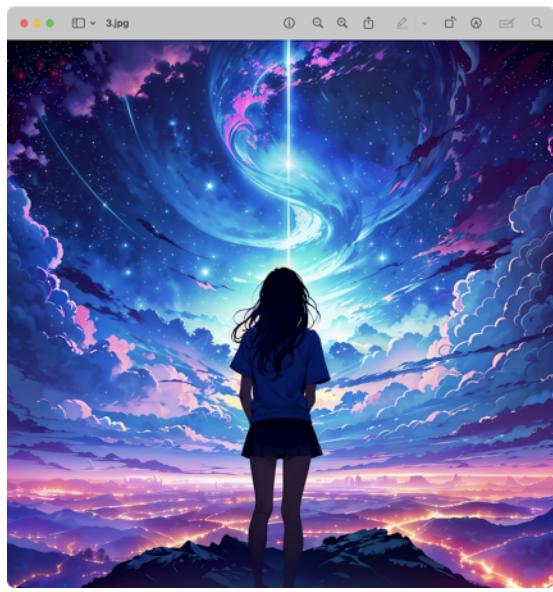
1.



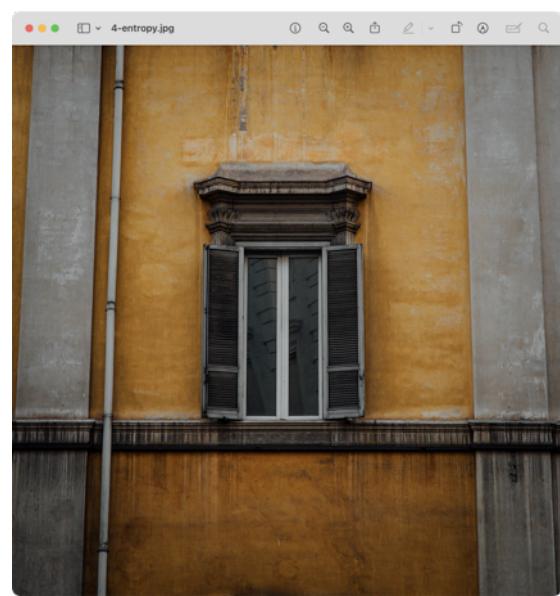
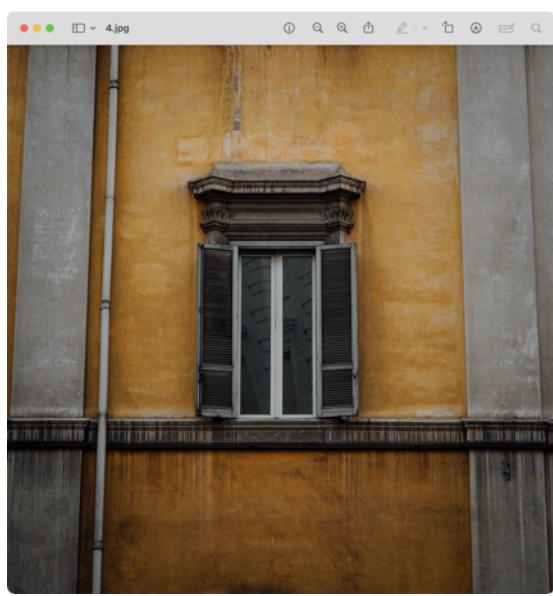
2.



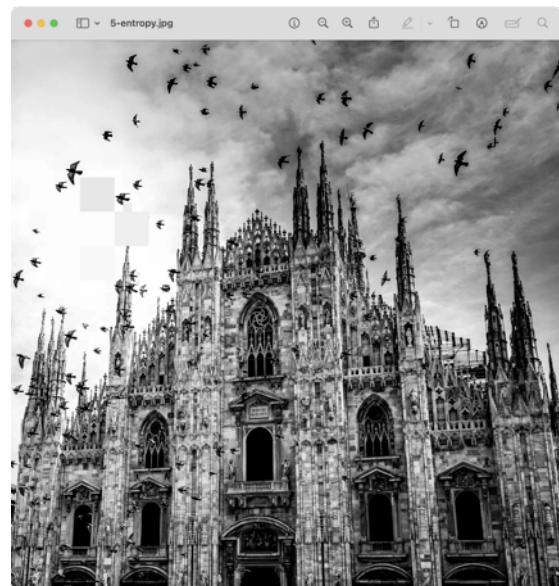
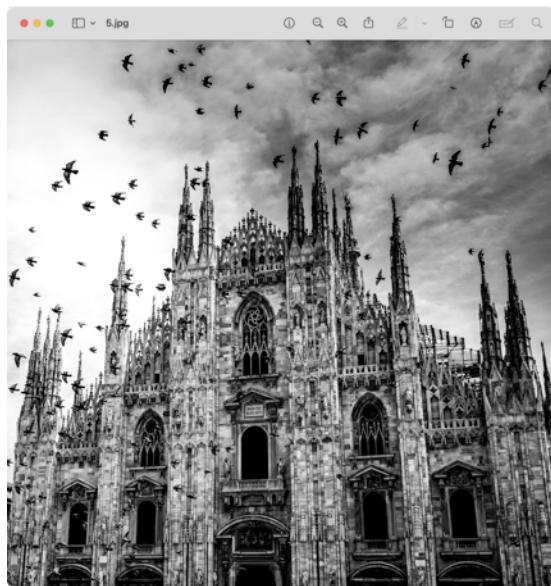
3.



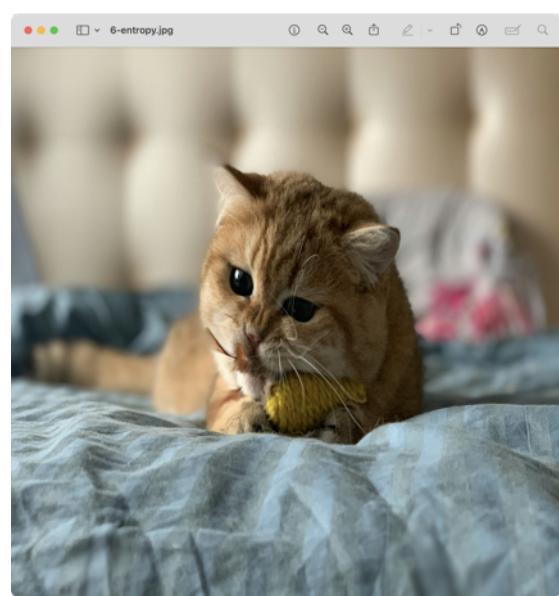
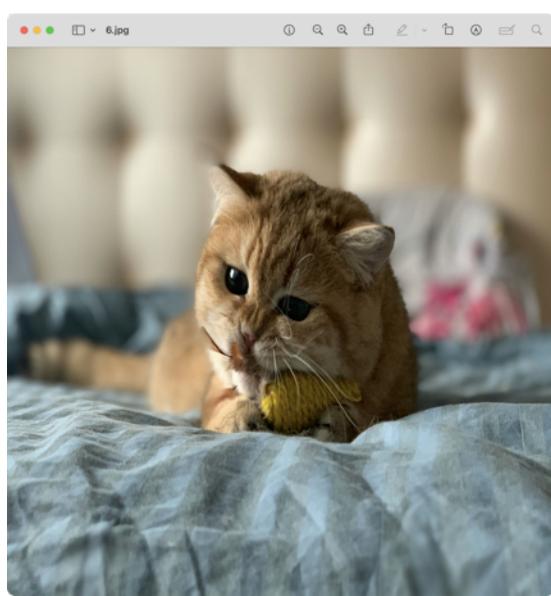
4.

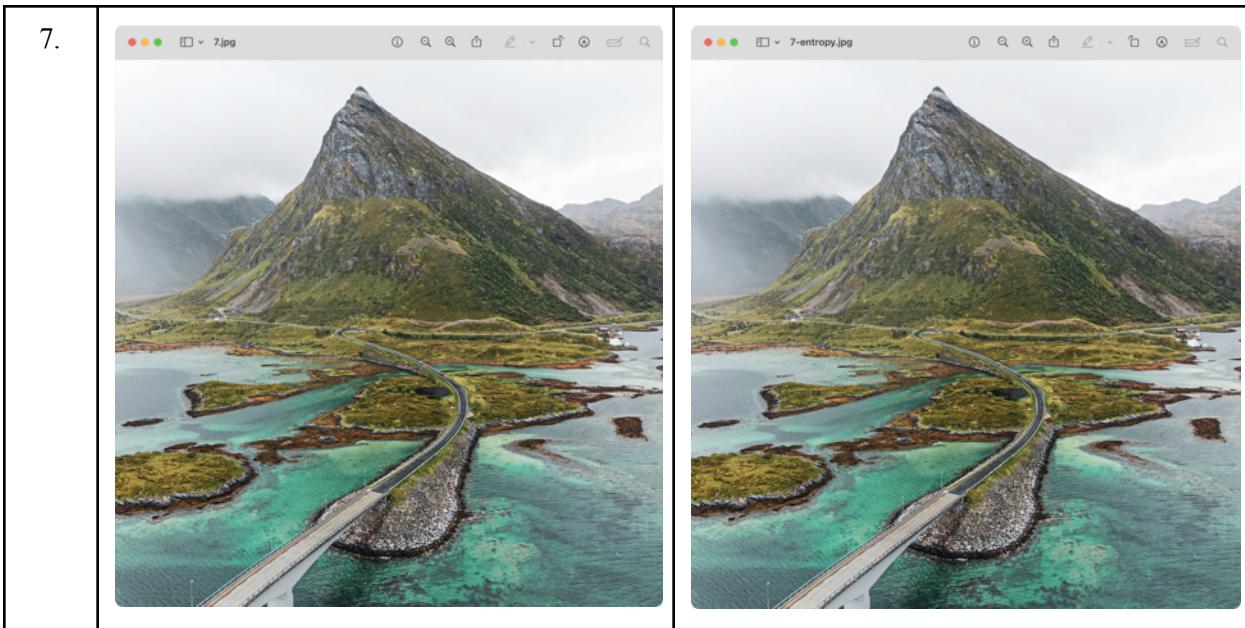


5.



6.





No.	Ukuran Awal (KB)	Ukuran Akhir (KB)	Kompresi (%)
1	942	261	72,27
2	139	64	53,85
3	1277	502	60,64
4	2669	1551	41,90
5	2057	2085	-1,39
6	1263	682	46,00
7	1955	1273	34,87

4.2.5 Error Handling

Beberapa *error handling* yang diatasi dalam program penulis yaitu kesalahan dalam memasukkan email absolut, ketidaksesuaian format, dan input yang diluar batas seharusnya.

1. Alamat absolut gambar

Jika pengguna berusaha memasukkan alamat absolut yang tidak valid atau tidak ditemukan dalam device, program akan mengeluarkan pesan error.

2. Pilihan Metode Pengukuran Error

Ketika pengguna memasukkan integer di luar 1 dan 4 maupun tipe data lain selain integer, program akan mengeluarkan pesan error.

3. Input Nilai Threshold

Ketika threshold di bawah nol, program akan mengeluarkan pesan error.

4. Input Ukuran Blok Minimum

Ketika ukuran blok minimum di bawah nol, program akan mengeluarkan pesan error.

```
Tucil2_13523015_13523029 — java -cp bin Main — 80x24
[(base) mac@Kaindras-MacBook-Pro Tucil2_13523015_13523029 % java -cp bin Main ]
```

Welcome to, Quadtree Image Compressor
A Project by Kaindra and Ho

```
Masukkan alamat absolut gambar: wrong-address
Alamat tidak dapat dibaca.

Masukkan alamat absolut gambar:
```

```
● ○ ● Tucil2_13523015_13523029 — java -cp bin Main — 80x24
[(base) mac@Kaindras-MacBook-Pro Tucil2_13523015_13523029 % java -cp bin Main ]
```

Welcome to, Quadtree Image Compressor
A Project by Kaindra and Ho

Masukkan alamat absolut gambar: wrong-address
Alamat tidak dapat dibaca.

Masukkan alamat absolut gambar: /Users/mac/Downloads/Tucil2_13523015_13523029/do
c/test-cases/3.jpg

Pilih Metode Pengukuran Error

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy

Masukkan pilihan (1 sampai 4): -1
Input invalid, input harus berupa integer pada range 1 sampai 4.

Masukkan pilihan (1 sampai 4):

```
● ○ ● Tucil2_13523015_13523029 — java -cp bin Main — 80x24
```

Welcome to, Quadtree Image Compressor
A Project by Kaindra and Ho

Masukkan alamat absolut gambar: wrong-address
Alamat tidak dapat dibaca.

Masukkan alamat absolut gambar: /Users/mac/Downloads/Tucil2_13523015_13523029/do
c/test-cases/3.jpg

Pilih Metode Pengukuran Error

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy

Masukkan pilihan (1 sampai 4): -1
Input invalid, input harus berupa integer pada range 1 sampai 4.

Masukkan pilihan (1 sampai 4): tipe-data-lain
Masukkan nilai integer yang valid.

Masukkan pilihan (1 sampai 4):

● ○ ● Tucil2_13523015_13523029 — java -cp bin Main — 80x24

Alamat tidak dapat dibaca.

Masukkan alamat absolut gambar: /Users/mac/Downloads/Tucil2_13523015_13523029/do
c/test-cases/3.jpg

Pilih Metode Pengukuran Error

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy

Masukkan pilihan (1 sampai 4): -1

Input invalid, input harus berupa integer pada range 1 sampai 4.

Masukkan pilihan (1 sampai 4): tipe-data-lain

Masukkan nilai integer yang valid.

Masukkan pilihan (1 sampai 4): 1

Masukkan threshold error: -1

Threshold tidak boleh negatif.

Masukkan threshold error: ■

● ○ ● Tucil2_13523015_13523029 — java -cp bin Main — 80x24

Welcome to, Quadtree Image Compressor
A Project by Kaindra and Ho

Masukkan alamat absolut gambar: /Users/mac/Downloads/Tucil2_13523015_13523029/do
c/test-cases/3.jpg

Pilih Metode Pengukuran Error

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy

Masukkan pilihan (1 sampai 4): 1

Masukkan threshold error: 15

Masukkan ukuran blok minimum: -1

Input harus lebih dari nol dan berupa integer yang valid.

Masukkan ukuran blok minimum: ■

5 Kesimpulan dan Saran

5.1 Kesimpulan

Melalui penggeraan tugas kecil ini, penulis telah berhasil mengimplementasikan algoritma *divide and conquer* dalam kompresi gambar dengan metode Quadtree. Setiap metode pengukuran error memiliki kelebihan dan kekurangannya masing-masing. Berdasarkan percobaan yang dilakukan, Kompresi paling efisien dan efektif (dengan kualitas yang cenderung masih mirip) adalah dengan menggunakan metode perhitungan Variance dan Entropy dengan detail sebagai berikut.

1. Variance menghasilkan struktur quadtree yang cenderung rata dan mampu mempertahankan detail dengan baik. Metode ini secara memberikan rasio kompresi tinggi dengan kualitas gambar yang masih terjaga. Namun, metode ini tidak terlalu optimal untuk menangani transisi warna gradien yang halus.
2. Entropy memiliki performa yang mirip dengan Variance dalam melakukan. Metode ini menghasilkan *result* yang halus dan tetap detail, meskipun terdapat background gradient. Meskipun begitu, waktu eksekusinya lebih tinggi, terutama untuk gambar berukuran besar.

MAD dan Max Pixel Difference menawarkan pilihan untuk melakukan kompresi yang lebih ringan dan cepat, meskipun tidak sebagus kedua metode di atas.

5.2 Saran

Dalam penggeraan tugas kecil ini, terdapat beberapa saran yang akan kami perhatikan untuk dikembangkan kedepannya, antara lain:

1. Mengerjakan tugas yang diberikan tanpa menunda waktu agar hasil bisa lebih maksimal.
2. Membuat program yang lebih interaktif dan *user friendly*. Misalnya dengan menambahkan fitur *drag and drop* untuk input file, sehingga *user* tidak perlu mengetik atau menyalin *path file* secara manual.
3. Menambah referensi dengan menelusuri internet ataupun berdiskusi dengan teman untuk memastikan konsep tidak salah dimengerti.

6 Lampiran

6.1 Pranala Github

Repositori Github untuk proyek ini dapat diakses dengan menekan tulisan ini atau dengan mengunjungi pranala https://github.com/MaheswaraKaindra/Tucil2_13523015_13523029.git.

6.2 Tabel Evaluasi Program

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	✓	

7 Referensi

Munir, R. (2025). Algoritma Divide and Conquer (Bagian 1). Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[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)

York, T. (2020). Quadtrees for Image Compression.

<https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>