



Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

TESIS DE LICENCIATURA

Segmentación de imágenes por color
utilizando el algoritmo de k-means clustering:
Implementación y compresión de datos

Presenta:

Brian Alberto Moreno Ortega

Para obtener el grado de

Licenciado en Matemáticas Aplicadas

Directores de Tesis:

M.C. Sergio Adán Juárez

Dr. José Rubén Conde Sánchez

Página intencionalmente en blanco

Agradecimientos

A mi papá, quien con su amor y apoyo incondicional ha sido fundamental en el logro de cada una de mis metas y sueños.

A mi mamá, un pilar en mi vida, que siempre está presente para mí, guiándome y enseñándome con amor para que me convierta en una mejor persona.

A mi hermana, mi mejor amiga y compañera de vida, que ha estado a mi lado en cada paso del camino.

A mi novia, Brenda, cuyo amor y compañía me fortalecen y animan a seguir adelante.

Expreso mi profundo agradecimiento a mi asesor de tesis, el M.C. Sergio Adán Juárez, por sus valiosos consejos y la ayuda brindada durante el desarrollo de este trabajo.

Igualmente al Dr. José Rubén Conde Sánchez por sus valiosas contribuciones.

A mi mejor amigo, Oscar, quien ha sido mi cómplice durante toda la carrera, compartiendo momentos inolvidables y brindándome su apoyo constante.

A mi amigo, Lalo, por su apoyo incondicional, su escucha atenta y los grandes momentos compartidos juntos.

A mi amiga, Sofía, quien comprende a la perfección los desafíos y sacrificios que hemos enfrentado para culminar este proyecto.

A todos ustedes, quiero expresar mi más sincero agradecimiento. Sus contribuciones, apoyo y amistad han sido invaluable en este camino hacia la culminación de esta tesis.

Página intencionalmente en blanco

Resumen

Esta tesis se enfoca en el desarrollo de un método para la segmentación de imágenes por color utilizando el algoritmo de k-means clustering. Se implementó un programa en lenguaje C que permite realizar la segmentación de imágenes con base en diferentes cantidades de clusters. Previo a la aplicación del algoritmo de k-means, se llevó a cabo una reducción del volumen de datos mediante la eliminación de filas y columnas redundantes, lo cual también contribuyó a la compresión de la imagen. Asimismo, se aplicó el método de compresión de datos RLE para almacenar las imágenes segmentadas. Se realizaron pruebas con diferentes cantidades de clusters y se evaluó la calidad de la segmentación obtenida. Los resultados demostraron que el algoritmo de k-means clustering implementado en lenguaje C fue capaz de segmentar correctamente las imágenes, preservando y resaltando los objetos de interés. Además, se observó que la eliminación de filas y columnas redundantes permitió una reducción significativa del tamaño de los datos, y la compresión por RLE resultó en una disminución adicional del tamaño de la imagen. En los anexos se incluyen los códigos en lenguaje C utilizados para la segmentación por k-means clustering, la compresión y la descompresión del archivo comprimido RLE. Esta tesis proporciona una base para la comprensión y aplicación de la segmentación de imágenes por color, también un enfoque práctico para la compresión de datos en el contexto de este proceso.

Contenido

Contenido	I
Lista de figuras	III
Lista de tablas	V
1. Introducción	1
1.1. Motivación	1
1.2. Justificación	2
1.3. Objetivos	2
1.3.1. Objetivo general	2
1.3.2. Objetivos particulares	2
1.4. Estructura de la tesis	3
2. Antecedentes	5
2.1. Segmentación de imágenes por color	5
2.2. Introducción al algoritmo de k-means clustering	5
2.3. Trabajos previos de k-means clustering con otras herramientas	6
2.4. Implementación en lenguaje C	6
3. Fundamentos teóricos	7
3.1. Imagen de mapa de bits	7
3.2. Imágenes BMP exactas e inexactas	8
3.3. Cálculo del tamaño de una imagen	10
3.4. Modelo RGB	11
3.5. Compresión de datos	12

3.5.1. Eliminación de código redundante	13
3.5.2. Eliminación de pixeles redundantes	13
3.5.3. Eliminación de redundancia psicovisual	13
3.6. Algoritmo de Compresión RLE	14
3.7. Histograma de niveles de gris	15
3.8. Histograma por fila de una imagen	16
3.9. Histograma por columna de una imagen	18
3.10. Clustering	18
3.11. K-Means Clustering	19
4. Metodología	25
4.1. Compresión por eliminación de redundancia psicovisual y pixeles redundantes.	25
4.2. Implementación en lenguaje C del algoritmo k-means clustering para la segmentación de imágenes por color.	27
4.3. Parametros de entrada	28
5. Análisis y resultados	29
5.1. Jirafa segmentada con 3 clusters	29
5.2. Jirafa segmentada con 10 clusters	29
5.3. Jirafa segmentada con 50 clusters	30
5.4. Jirafa segmentada con 100 clusters	31
5.5. Jirafa segmentada con 1000 clusters	31
5.6. Comparativa entre clusters	33
5.7. Pruebas con diferentes imágenes	34
5.8. Almacenamiento de los resultados con RLE	34
Conclusiones	37
Referencias	39
Código en Lenguaje C	41
.1. Programa de segmentación por k-means clustering	41
.2. Programa de compresión por RLE	46
.3. Programa de descompresión de archivos RLE	48

Lista de figuras

3.1. Bloques de una imagen en mapa de bits.	8
3.2. Convenio de ejes de una imagen BMP.	9
3.3. Píxeles en una imagen de mapa de bits.	9
3.4. Equivalencia entre un bit y un byte.	10
3.5. Imagen de un girasol con dimensiones de 1368x906 píxeles.	11
3.6. Adición de los colores primarios de la luz.	11
3.7. Cubo de valores RGB.	12
3.8. Imagen de 8×8 píxeles.	15
3.9. Histograma de la Figura 3.8.	16
3.10. Imagen con dimensiones 1600x1200 píxeles.	17
3.11. Histograma del canal verde de las filas 674 y 675 de la Figura 3.10.	17
3.12. Histograma del canal azul de las filas 674 y 675 de la Figura 3.10.	17
3.13. Histograma del canal rojo de las filas 674 y 675 de la Figura 3.10.	18
3.14. Puntos A y B.	19
3.15. Conjunto de puntos.	20
3.16. Grupos	21
3.17. Paso 1 del algoritmo de k-means clustering.	21
3.18. Paso 2 del algoritmo de k-means clustering.	22
3.19. Paso 3 del algoritmo de k-means clustering.	22
4.1. Diagrama de la metodología de este trabajo.	25
4.2. Imagen de original sin filas pares de píxeles.	26
4.3. Imagen de la Figura 3.10 sin filas ni columnas pares.	26
4.4. Comparación de imágenes despues de la Fase I.	27

5.1. Jirafa segmentada con 3 clusters.	30
5.2. Jirafa segmentada con 10 clusters.	30
5.3. Jirafa segmentada con 50 clusters.	31
5.4. Jirafa segmentada con 100 clusters.	32
5.5. Jirafa segmentada con 1000 clusters.	32
5.6. Comparación de los resultados con diferentes clusters	33
5.7. Comparación de Lena con 10 clusters.	34
5.8. Comparación de aguacates con 10 clusters.	35
5.9. Comparación de platos de talavera con 10 clusters.	35

Lista de tablas

3.1. Niveles de gris posibles en un pixel.	8
3.2. Contenido de los primeros 54 bytes de la cabecera de una imagen BMP.	9
3.3. Bytes extra por fila en imágenes inexactas.	10
3.4. Niveles de gris posibles de la Figura 3.8.	15
3.5. Media de los puntos A, B y C.	20
3.6. Distancias euclidianas a los centroides c_1 y c_2	21
4.1. Equipo utilizado para las pruebas.	28
5.1. Tamaño en bytes después de comprimir con RLE	36

Página intencionalmente en blanco

Glosario

C

Clustering

El clustering es un método que tiene como finalidad principal lograr el agrupamiento de conjuntos de objetos no etiquetados, para lograr construir subconjuntos de datos conocidos como Clusters.

5

Codificación de longitud de ejecución (RLE)

Es un algoritmo de compresión de datos en el que las secuencias de datos con el mismo valor consecutivas son almacenadas como un único valor más su recuento. 14

Compresión

Reducción del volumen de datos tratables para representar una determinada información empleando una menor cantidad de espacio. 14

D

Datos

Son una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable cuantitativa o cualitativa. Los datos aisladamente pueden no contener información humanamente relevante. Solo cuando un conjunto de datos se examina conjuntamente a la luz de un enfoque, hipótesis o teoría se aprecia la información contenida en dichos datos. 12

I

Información

Es el nombre asociado a un conjunto organizado de datos procesados que constituyen un mensaje que cambia el estado de conocimiento del sujeto o sistema que recibe el mensaje. 12

M

Mapa de bits

Se conoce como mapa de bits a un formato que sirve para representar imágenes de forma digital, en el que las dimensiones están determinadas por la cantidad de píxeles horizontales y verticales que componen a cada imagen, y en el que cada pixel tiene asociado un rango de valores de color. 12

R

Redundancia

Se refiere a la información que se expresa más de una vez y su reiteración permite, pese a la ausencia de esta, rearmar su contenido. 13

Resolución

Cantidad de detalles que puede observarse en una imagen (numero de pixeles por pulgada) entre mayor cantidad de píxeles por pulgada, mayor detalle en la imagen. 8

S

Segmentación

Se conoce como segmentación al acto de dividir una imagen digital en varias regiones (grupos de píxeles) denominadas segmentos.. 5

Acrónimos

B

BMP

Mapa de Bits. 8

R

RGB

Rojo, Verde y Azul. 11

RLE

Codificación de Longitud de Ejecución. 14

Página intencionalmente en blanco

Capítulo 1

Introducción

En el entorno cada vez más visual y tecnológico en el que nos encontramos inmersos, la necesidad de procesar y comprender imágenes de manera eficiente y precisa se ha vuelto imperativa. La segmentación de imágenes por color surge como un desafío fundamental en este contexto, ya que su capacidad para discernir y aislar regiones de interés no solo tiene un impacto directo en campos como la medicina, la industria y la investigación científica, sino que también influye en la toma de decisiones informadas. La identificación y extracción de objetos relevantes de un fondo visual confuso resulta crucial para la comprensión y análisis adecuados de la información visual. Además, el constante aumento en la generación de datos visuales exige soluciones que no solo garanticen la precisión en la segmentación, sino también la optimización del almacenamiento y la transmisión de estas imágenes. Por ende, esta tesis se erige en respuesta a la imperante necesidad de desarrollar un método eficaz y certero para la segmentación por color, con el algoritmo de k-means clustering como necesidad.

1.1. Motivación

La segmentación de imágenes por color es un desafío clave en el procesamiento de imágenes y tiene aplicaciones significativas en diversas áreas, como la medicina, la industria y la investigación científica. La capacidad de identificar y aislar objetos de interés en imágenes aporta información valiosa para el análisis y la toma de decisiones. Además, la necesidad de reducir el volumen de datos y mejorar la eficiencia de almacenamiento es una preocupación constante en el procesamiento de imágenes. En este contexto, desarrollar un método eficiente y preciso para la segmentación por color utilizando el algoritmo de k-means clustering podría tener un impacto considerable en la mejora de diversas aplicaciones que requieren análisis de imágenes.

1.2. Justificación

La elección de desarrollar un método de segmentación de imágenes por color utilizando el algoritmo de K-means clustering se justifica por diversas razones interrelacionadas. En primer lugar, la segmentación de imágenes es un problema esencial en el ámbito del procesamiento de imágenes y posee aplicaciones de relevancia en áreas como la medicina, la industria y la investigación científica. La acción de identificar y aislar objetos de interés en imágenes constituye un aporte valioso para el análisis y la toma de decisiones informadas. Adicionalmente, la necesidad inherente de reducir la magnitud de los datos y mejorar la eficiencia del almacenamiento representa un reto constante en el procesamiento de imágenes. En este contexto, la creación de un método eficiente y preciso para la segmentación por color, a través de la implementación del algoritmo de k-means clustering, podría resultar en un impacto importante en la mejora de diversas aplicaciones que requieren un análisis más profundo de las imágenes. Asimismo, esta tesis contribuye al proporcionar una implementación detallada y códigos en lenguaje C, lo que facilita la replicación de los resultados y se convierte en un recurso valioso para otros investigadores y profesionales que deseen explorar la segmentación de imágenes y la compresión de datos.

1.3. Objetivos

1.3.1. Objetivo general

Implementar el algoritmo de k-means clustering para la segmentación por color en imágenes con formato de mapa de bits (BMP).

1.3.2. Objetivos particulares

1. Realizar una revisión de la literatura sobre algoritmos de segmentación de imágenes y técnicas de clustering, con énfasis en el algoritmo de k-means y sus variantes.
2. Seleccionar y preparar adecuadamente un conjunto diverso de imágenes en formato BMP para ser utilizado como base de prueba en la implementación del algoritmo de k-means.
3. Implementar y optimizar el algoritmo de k-means clustering para su aplicación en la segmentación por color de imágenes BMP.

El algoritmo de k-means clustering, ampliamente utilizado para agrupar datos según su similitud, se empleará en la segmentación de imágenes para agrupar píxeles similares en segmentos representativos

basados en sus valores de color RGB. Además de su implementación, se investigarán técnicas de compresión de datos con el fin de reducir el volumen de las imágenes segmentadas y evaluar los resultados obtenidos.

1.4. Estructura de la tesis

La tesis es organizada de la siguiente manera:

- Capítulo 1: En este capítulo se brinda una visión general del tema de la segmentación de imágenes por color y se establece la motivación, justificación y objetivos.
- Capítulo 2: Se presentan los antecedentes teóricos y prácticos relacionados con la segmentación de imágenes por color utilizando el algoritmo de k-means clustering. Se revisan trabajos previos relevantes que respaldan la elección de implementar este algoritmo en lenguaje C para la segmentación de imágenes por color.
- Capítulo 3: Se abordan los fundamentos teóricos y los conceptos clave relacionados con la segmentación de imágenes, el formato BMP, la compresión de datos en imágenes, el histograma de una imagen, la eliminación de filas y columnas redundantes.
- Capítulo 4: El clustering y el algoritmo de k-means clustering son abordados en este capítulo. Se describe la metodología utilizada para implementar el algoritmo de k-means clustering y llevar a cabo la segmentación de imágenes por color en imágenes BMP. Se explican los pasos específicos, los criterios de ejecución y los aspectos técnicos relevantes.
- Capítulo 5: Se presentan y analizan los resultados obtenidos a partir de la implementación del algoritmo de K-means clustering en la segmentación de imágenes por color. Se evalúa la calidad de la segmentación y se discuten las ventajas y limitaciones del enfoque propuesto.
- Capítulo 6: Se resumen las conclusiones principales de la investigación y sus contribuciones.

La estructura de la tesis explora los fundamentos teóricos, presenta la metodología propuesta, analiza y logra resultados sobre la eficacia del enfoque propuesto. Se espera contribuir al campo de la segmentación de imágenes por color y demostrar la viabilidad del algoritmo de k-means clustering para esta tarea.

Página intencionalmente en blanco

Capítulo 2

Antecedentes

2.1. Segmentación de imágenes por color

La segmentación de imágenes por color es una técnica utilizada en el procesamiento de imágenes para dividir una imagen en regiones o segmentos basados en su contenido de color. Esta técnica es especialmente útil en aplicaciones donde la información cromática es relevante, como el análisis de imágenes médicas, reconocimiento de objetos o detección de áreas de interés [Palomino and Concha, 2009].

2.2. Introducción al algoritmo de k-means clustering

El algoritmo de k-means clustering es una técnica popular y ampliamente utilizada en el campo de la minería de datos y el procesamiento de imágenes para la agrupación de datos y la segmentación de imágenes. Este algoritmo busca agrupar un conjunto de datos en k grupos o clusters de manera que los elementos dentro de cada grupo sean similares entre sí y distintos de los elementos en otros grupos [Sanz, 2023].

Para la segmentación por color, el algoritmo de k-means clustering agrupa los píxeles de una imagen en k grupos distintos, donde k representa el número de colores en los que se segmenta la imagen. El objetivo es asignar cada píxel al grupo que tenga el valor de color más cercano.

2.3. Trabajos previos de k-means clustering con otras herramientas

Varios trabajos previos han utilizado el algoritmo de k-means clustering para la segmentación de imágenes por color utilizando distintas herramientas. Estos trabajos demuestran la efectividad y versatilidad del algoritmo en diferentes entornos y aplicaciones. Algunos ejemplos significativos son los siguientes:

- En el trabajo de [Doe et al., 2019], se aplicó el algoritmo de k-means clustering en el campo de la medicina para segmentar imágenes médicas en aplicaciones de diagnóstico. Utilizaron una implementación del algoritmo en el entorno de MATLAB, logrando resultados precisos y confiables en la segmentación por color.
- En otro estudio realizado por [Rodríguez et al., 2017], se utilizó el algoritmo de k-means clustering en el procesamiento de imágenes satelitales para la detección de objetos en áreas urbanas. Implementaron el algoritmo en el lenguaje Python junto con bibliotecas como NumPy y scikit-learn, logrando una segmentación eficiente y precisa.

Estos trabajos previos demuestran la utilidad y la aplicabilidad del algoritmo de k-means clustering en la segmentación de imágenes por color con distintas herramientas de programación. Sin embargo, a pesar de los resultados obtenidos en estos trabajos, es importante considerar las ventajas y desventajas de cada herramienta de programación al elegir el lenguaje para implementar el algoritmo de k-means clustering.

2.4. Implementación en lenguaje C

El lenguaje C ofrece diversas ventajas que lo hacen adecuado para esta tarea:

- **Eficiencia:** El lenguaje C es conocido por su eficiencia en el uso de recursos y la velocidad de ejecución. Dado que la segmentación de imágenes por color implica procesar grandes volúmenes de datos y realizar cálculos intensivos, el rendimiento eficiente es esencial para obtener resultados en tiempo real.
- **Portabilidad:** El lenguaje C es compatible y portátil en diferentes plataformas y sistemas operativos. Esto asegura que el programa desarrollado pueda ejecutarse en una variedad de entornos sin problemas de compatibilidad.

Los factores como la eficiencia y la portabilidad son clave para la elección del lenguaje C como herramienta de programación para la implementación de la segmentación de imágenes por color utilizando el algoritmo de k-means clustering.

Capítulo 3

Fundamentos teóricos

En este capítulo, se presentarán los fundamentos teóricos necesarios para comprender el contexto y las técnicas utilizadas en la implementación del algoritmo de segmentación de imágenes por color utilizando el método de k-means clustering. Se abordarán conceptos clave relacionados con la estructura de las imágenes en formato BMP, el modelo de color RGB, la compresión de datos, técnicas de eliminación de redundancia e histogramas de nivel de gris.

3.1. Imagen de mapa de bits

Una imagen digital se puede definir como una función bidimensional discreta de intensidad de luz $I(x, y)$, donde (x, y) representan las coordenadas espaciales y el valor I en un punto cualquiera (x, y) , es proporcional al nivel de gris de la imagen en ese punto.

Las imágenes digitales también se consideran una matriz, cuyos índices de fila y columna identifican un punto de la imagen y el valor del correspondiente elemento de la matriz indica el nivel de gris en ése punto [González and Woods, 1999].

A los elementos de este tipo de distribución digital se les denominan píxeles y cada uno de ellos usando el modelo RGB, está representado por 24 bits y se le puede asignar uno de los 16,777,216 colores diferentes (2^{24}). Este formato se puede considerar como universal ya que prácticamente todas las aplicaciones lo pueden manejar [Cruz Contreras et al., 2004].

En la siguiente tabla se muestra el número de colores que se le pueden asignar a un píxel dependiendo la cantidad de bits usados para representarlo.

Bits por pixel	Número de niveles de gris que se le pueden asignar a un pixel
1	$2^1 = 2$
2	$2^2 = 4$
4	$2^4 = 16$
8	$2^8 = 256$
16	$2^{16} = 65,536$
24	$2^{24} = 16,777,216$

Tabla 3.1: Niveles de gris posibles en un pixel.

Una imagen de mapa de bits está formada por dos bloques; el primero es la cabecera; el segundo son los datos propios de la imagen, Figura 3.1.

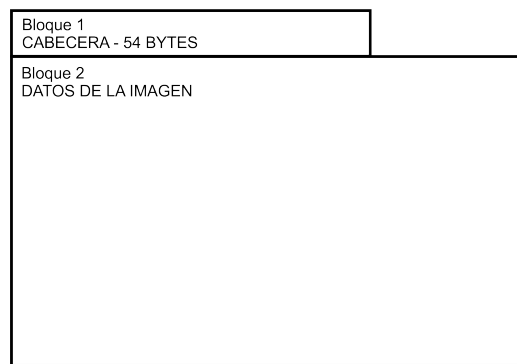


Figura 3.1: Bloques de una imagen en mapa de bits.

La cabecera de las imágenes en mapa de bits comienza con las letras BM(0x42 0x4D), esto sirve para identificarla con el programa de visualización o edición. En la cabecera se encuentra información de la imagen, como el tamaño del archivo, la resolución, sus dimensiones, la cantidad de bytes que representa el color de cada pixel, etc. En la Tabla 3.2 se muestra el contenido de cada uno de los 54 bytes que componen la cabecera.

El segundo bloque de una imagen BMP contiene los datos de la misma. Como se muestra en la Figura 3.3, comienzan a leerse de abajo hacia arriba y de izquierda a derecha, partiendo de la coordenada $I(0, 0)$, donde la primera componente X representa las filas y Y las columnas, es decir, en una imagen de 24 bits, los primeros 3 bytes corresponden al color del primer pixel inferior izquierdo.

3.2. Imágenes BMP exactas e inexactas

En las Imágenes de Mapa de Bits (BMP) hay que considerar dos tipos: las imágenes exactas y las inexactas. Las exactas son aquellas en las que:

$$(W * P) \% 4 = 0$$

Bytes	Información del bitmap
0, 1	Tipo de fichero "BM"
2, 3, 4, 5	Tamaño del archivo
6, 7	Reservado
8, 9	Reservado
10, 11, 12, 13	Inicio de los datos de la imagen
14, 15, 16, 17	Tamaño de la cabecera del bitmap
18, 19, 20, 21	Anchura (Píxeles)
22, 23, 24, 25	Altura (Píxeles)
26, 27	Número de planos
28, 29	Tamaño de cada punto
30, 31, 32, 33	Compresión
34, 35, 36, 37	Tamaño de la imagen
38, 39, 40, 41	Resolución horizontal
42, 43, 44, 45	Resolución vertical
46, 47, 48, 49	Tamaño de la tabla de color
50, 51, 52, 53	Contador de contadores importantes

Tabla 3.2: Contenido de los primeros 54 bytes de la cabecera de una imagen BMP.



Figura 3.2: Convenio de ejes de una imagen BMP.

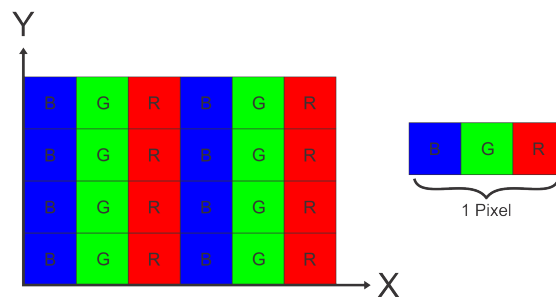


Figura 3.3: Píxeles en una imagen mapa de bits.

Donde W representa el valor del ancho de la imagen y P el peso de cada pixel.

Cuando el módulo es diferente a cero se dice que la imagen BMP es inexacta y dependiendo el valor del módulo se agregan bytes extras al final de cada fila.

La Tabla 3.3 muestra la cantidad de bytes que se agregan a cada una de las filas de la imagen:

Resultado del modulo	Cantidad de bytes extra por fila
0	3
1	2
2	1
3	0

Tabla 3.3: Bytes extra por fila en imágenes inexactas.

3.3. Cálculo del tamaño de una imagen

Para calcular el tamaño de una imagen digital se debe multiplicar la cantidad de pixeles de ancho por la cantidad de pixeles de alto que posee la imagen por el peso de cada pixel y al resultado se le suma la cabecera (54 bytes). El tamaño, conocido como peso, se expresa en bytes, KB o MB y no debe confundirse con el tamaño físico. El peso de cada pixel depende del modo de color. El modelo empleado es el RGB, por lo que se dividen los 24 bits entre 8, ya que 8 bits forman 1 byte (Figura 3.4). La imagen de la Figura 3.5 nos servirá para realizar un ejemplo.

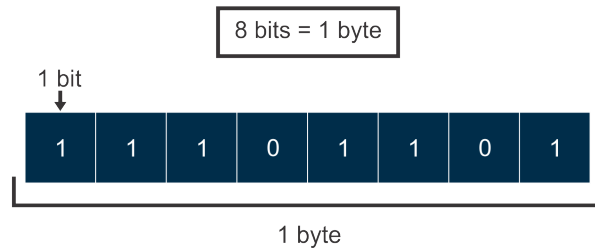


Figura 3.4: Equivalencia entre un bit y un byte.

Las imágenes empleadas son de 24 bits, es decir, 3 bytes, 1 byte por canal. Es importante mencionar que usaremos imágenes exactas. Para calcular el peso de la imagen de la Figura 3.5, implementamos la siguiente formula:

$$Peso = Ancho * PesoDelPixel * Alto + Cabecera$$

$$Peso = 1368 * 3 * 906 + 54$$

$$Peso = 3718278bytes.$$



Figura 3.5: Imagen de un girasol con dimensiones de 1368×906 píxeles.

3.4. Modelo RGB

Como se muestra en la Tabla 3.1, el número de colores que puede tomar cada píxel depende de la cantidad de bits que se utilizan para representarlo.

El modelo Rojo, Verde y Azul (RGB) es una síntesis aditiva que representa colores mediante la mezcla de sus 3 colores primarios de luz: rojo, verde y azul (Figura 3.6).

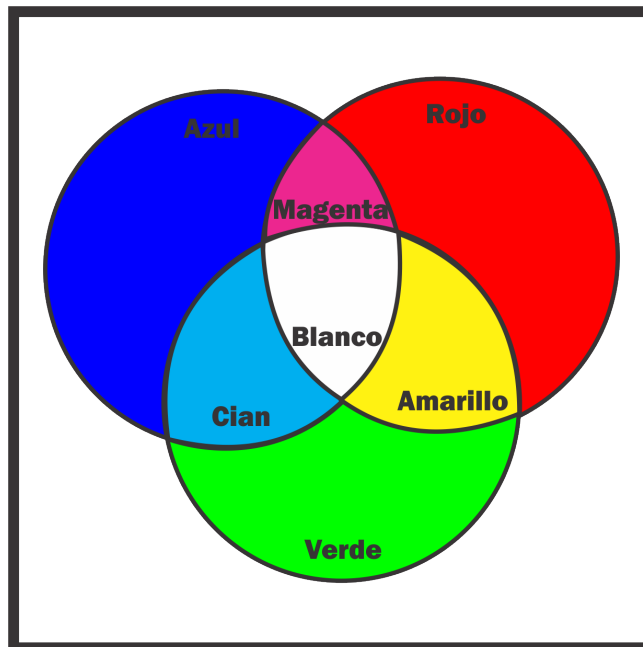


Figura 3.6: Adición de los colores primarios de la luz.

Este modelo es un sistema de coordenadas cartesianas. El subespacio de color de interés es el cubo de la Figura 3.7, en la que los valores RGB están en tres vértices; el cian, magenta y amarillo en otros tres

vértices; el negro en el origen; y el blanco en el vértice opuesto al origen [González and Woods, 1999].

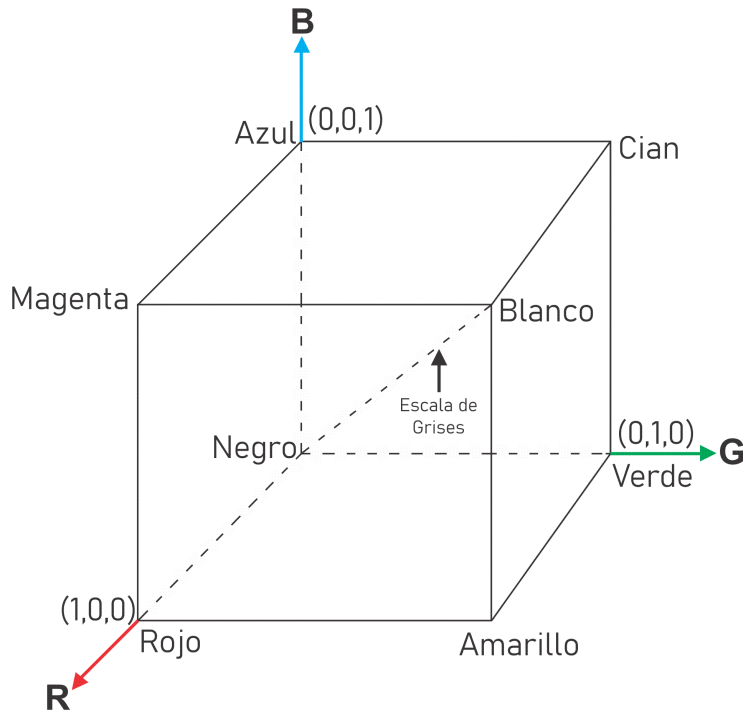


Figura 3.7: Cubo de valores RGB.

La escala de grises es una diagonal que se extiende del negro al blanco, los colores son puntos del cubo o de su interior. Por conveniencia, todos los valores de color han sido normalizados, de forma que el cubo de la Figura 3.7 es el cubo unidad, es decir todos los valores de R, G y B están en el intervalo $[0,1]$.

En un mapa de bits, cada color se codifica con un byte (8 bits). De esta forma, lo usual es que la intensidad de cada una de sus componentes RGB se mida en una escala de 0 a 255.

3.5. Compresión de datos

Las imágenes en mapas de bits tienen la desventaja de ocupar mucho espacio en archivos, ya que guardan los datos de color de cada píxel sin compresión. Aunque esto permite gráficos nítidos y de alta calidad, a menudo es necesario recurrir a la compresión para reducir el tamaño y agilizar su almacenamiento y procesamiento.

La compresión de datos es el proceso de reducción del volumen de datos para representar una determinada cantidad de Información. Es decir, un conjunto de datos puede contener datos redundantes

que son de poca relevancia o son datos que se repiten en el conjunto, los cuales si se identifican pueden ser eliminados [Lezama, 2017].

Es importante conocer la diferencia entre datos e información ya que no son sinónimos. “Los datos son los medios a través de los que se conduce la información. Se pueden utilizar distintas cantidades para describir la misma cantidad de información” [González and Woods, 1999].

A los datos que proporcionan información sin relevancia o repetitiva se les conoce como redundancia de datos y en la compresión digital de imágenes, se puede identificar 3 tipos básicos de redundancias:

3.5.1. Eliminación de código redundante

La eliminación de código redundante es una técnica de compresión que se basa en la identificación y eliminación de secuencias de código que se repiten de forma redundante en un archivo. En el contexto de imágenes, esto implica buscar patrones de píxeles o información codificada que se repiten en diferentes partes de la imagen. Al eliminar estos patrones redundantes y reemplazarlos por referencias más compactas, se logra reducir el tamaño del archivo de imagen sin perder detalles visuales importantes [Salomón, 2007].

3.5.2. Eliminación de píxeles redundantes

La eliminación de píxeles redundantes es una técnica de compresión que se centra en la identificación y eliminación de píxeles que no aportan información visual significativa a la imagen. Esto se logra mediante el análisis de la distribución de píxeles en la imagen y la identificación de áreas con patrones repetitivos o áreas vacías. Al eliminar estos píxeles redundantes, se reduce el tamaño del archivo de imagen sin comprometer la calidad visual [Wang et al., 2015].

3.5.3. Eliminación de redundancia psicovisual

La eliminación de redundancia psicovisual es una técnica de compresión que tiene en cuenta las características de la percepción visual humana para reducir el tamaño de los archivos de imagen. Esta técnica se basa en la comprensión de que los seres humanos tienen una menor sensibilidad a ciertos detalles visuales y pueden percibir cambios mínimos en la calidad de la imagen. Al aprovechar esta propiedad, se pueden eliminar ciertos detalles redundantes o menos perceptibles sin comprometer la calidad visual general de la imagen [Lee et al., 2015].

3.6. Algoritmo de Compresión RLE

El algoritmo de compresión de Codificación de Longitud de Ejecución (RLE) es un método de compresión de datos. La idea detrás de RLE es bastante simple: en lugar de almacenar cada valor individual en una secuencia de datos, el algoritmo comprime los datos almacenando los valores repetidos como un solo valor y la cantidad de veces que se repite. Por ejemplo, considerando la siguiente secuencia de caracteres:

AAAAABBCCCCDDAAA

Usando RLE, podemos comprimir esta secuencia almacenando las repeticiones como un solo valor y la cantidad de veces que se repite. La secuencia comprimida resultaría:

5A2B5C2D3A

Donde **5A** significa que la letra **A** se repite cinco veces, **2B** que la letra **B** se repite dos veces, y así sucesivamente. Es importante tener en cuenta el algoritmo de codificación de longitud de ejecución (RLE) funciona mejor en datos que tienen grandes secciones de valores repetidos. Si la secuencia de datos no tiene muchas repeticiones, entonces RLE no logra mucha compresión.

La compresión de una imagen mediante RLE, se basa en la observación de que, al seleccionar un píxel de la imagen al azar, existe una probabilidad muy alta de encontrar otros adyacentes a él con el mismo color [Salomón, 2007].

Para comprimir imágenes en escala de grises cada bloque de píxeles con la misma intensidad se codifica como un par (Cantidad de veces que se repite el píxel, Valor del píxel). La 'Cantidad de veces que se repite el píxel' normalmente ocupa 1 byte, lo que permite secuencias de hasta 255 píxeles.

Por ejemplo, un mapa de bits, en escala de grises de 8 bits de profundidad que comienza de esta forma:

100,100,100,100,100,100,100,100,100,50,50,50,255,0,0,...

Se comprime de la siguiente forma:

9,100,3,50,1,255,2,0,...

Donde los números subrayados son la cantidad de veces que se repite el siguiente elemento.

3.7. Histograma de niveles de gris

Para entender mejor el procesamiento en imágenes digitales es necesario comprender una herramienta útil: el histograma de niveles de gris. Es una herramienta gráfica que muestra las características de brillo de una imagen, es decir, proporciona una distribución gráfica de los niveles de gris de los píxeles en una imagen digital. Como ejemplo, usaremos la imagen que representa la Figura 3.8.

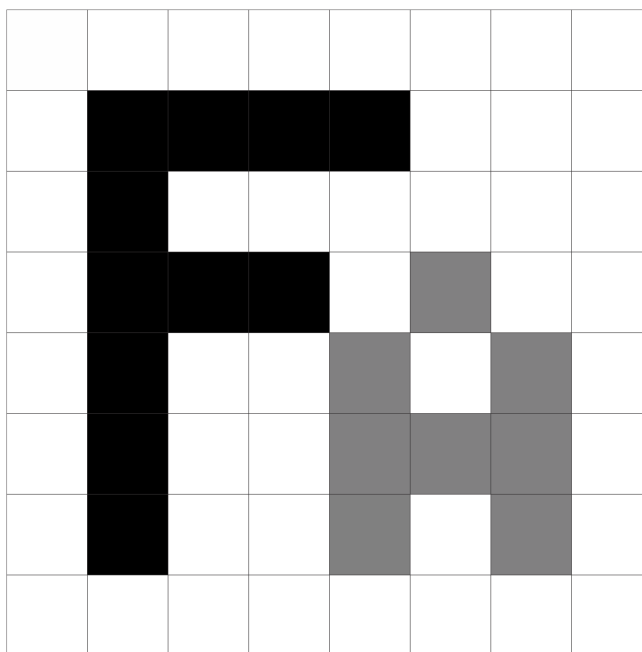


Figura 3.8: Imagen de 8×8 píxeles.

Se trata de una imagen con pocos niveles de gris y un tamaño de píxeles de 8×8 . Si usamos 2 bits para codificar el brillo de cada píxel, solo es posible obtener 4 niveles de gris (Tabla 3.1). En estos casos, los niveles de gris se numeran de 0 al 3, donde 0 corresponde al color negro y el 3 a un mayor brillo.

Nivel de gris	Brillo
0	Negro
1	Gris oscuro
2	Gris claro
3	Blanco

Tabla 3.4: Niveles de gris posibles de la Figura 3.8.

La gráfica que aparece en la Figura 3.9 corresponde al histograma de la imagen de la Figura 3.8.

Los números del eje X , corresponden a los niveles de gris que componen la imagen, a la izquierda encontraremos el valor más oscuro (negro) y al extremo derecho, el color más claro (blanco), mientras

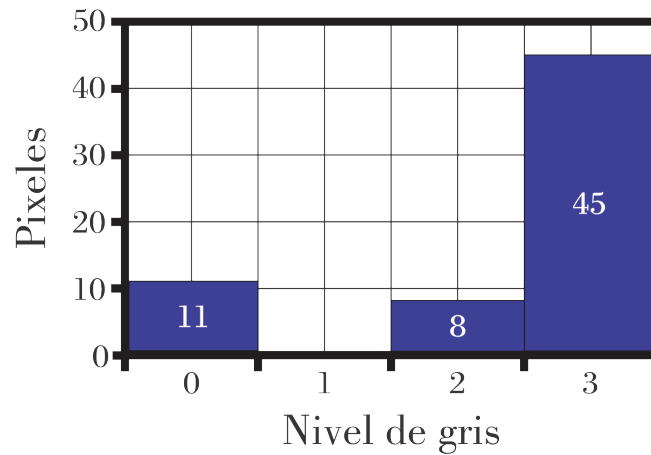


Figura 3.9: Histograma de la Figura 3.8.

que el resto de niveles se distribuyen de forma uniforme. En el eje Y está representada la cantidad de píxeles de la imagen con ese nivel de gris en específico. Se debe mencionar que en un histograma habitualmente no se encuentra enumerado el eje Y , ni la escala de tonos en el eje X . Usando el histograma podemos determinar que la imagen cuenta con 11 píxeles color negro (nivel 0), 8 con un tono gris claro (nivel 2), 45 píxeles completamente blancos (nivel 3) y ningún píxel con un tono gris oscuro (nivel 1).

3.8. Histograma por fila de una imagen

Al analizar el histograma de una imagen, podemos deducir varios detalles importantes que pueden ser de utilidad en distintas áreas, y también, como en este caso, para respaldar la eliminación de píxeles redundantes.

Para el siguiente ejemplo, graficaremos los histogramas por canal de las filas 674 y 675 de la Figura 3.10. El eje X corresponderá a la posición de los píxeles de las filas y el eje Y representará el nivel de gris de los píxeles en esa posición.

Este tipo de histogramas brindan información del nivel de gris de la zona de la que se está tomando la línea de píxeles. Al graficar dos filas en el mismo histograma podemos comparar el nivel de gris que hay entre píxeles vecinos. Se puede observar que los histogramas de cada fila tienen suficiente similitud en todo el ancho como para considerar que los niveles de gris entre filas contiguas son casi iguales.



Figura 3.10: Imagen con dimensiones 1600x1200 pixeles.

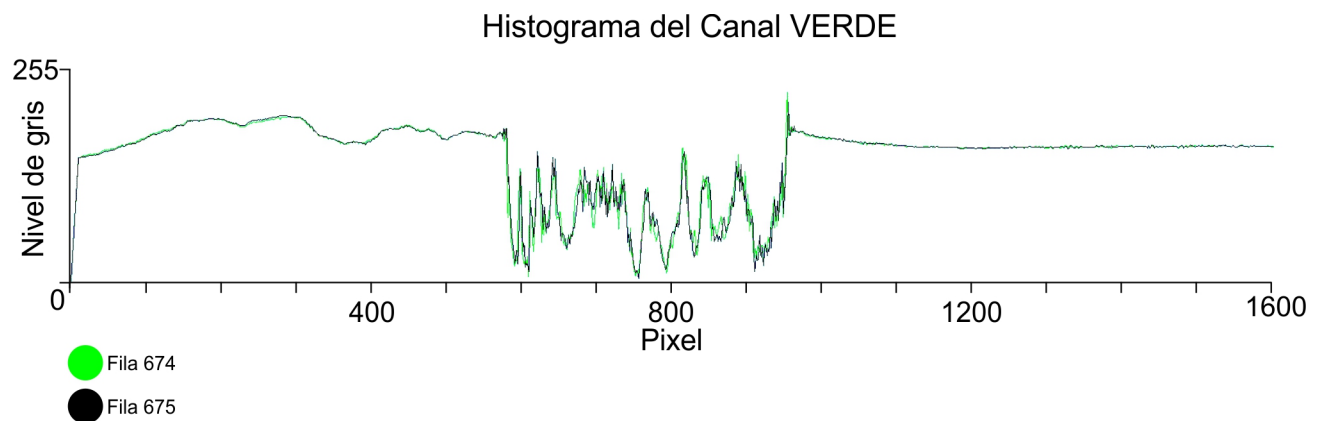


Figura 3.11: Histograma del canal verde de las filas 674 y 675 de la Figura 3.10.

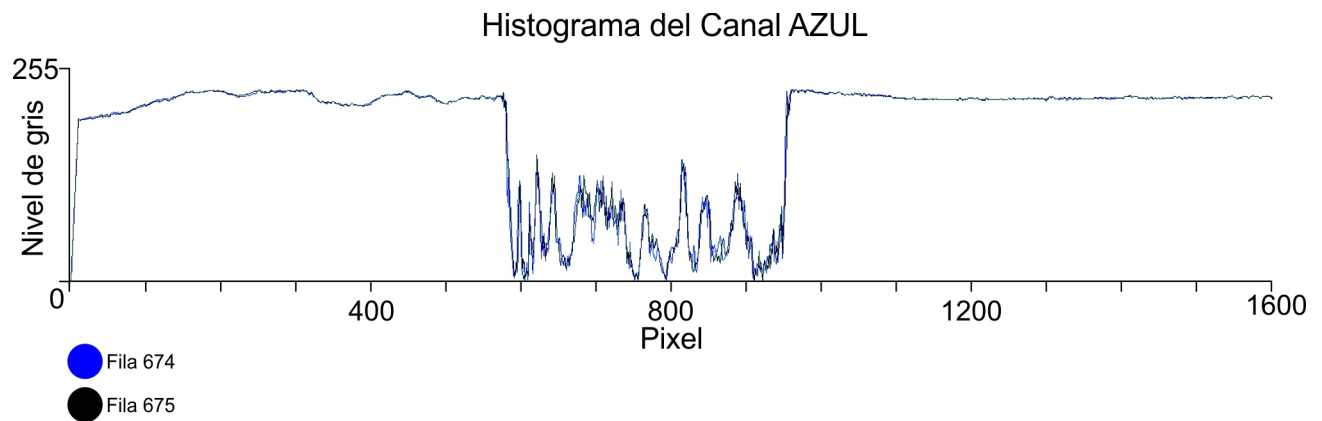


Figura 3.12: Histograma del canal azul de las filas 674 y 675 de la Figura 3.10.

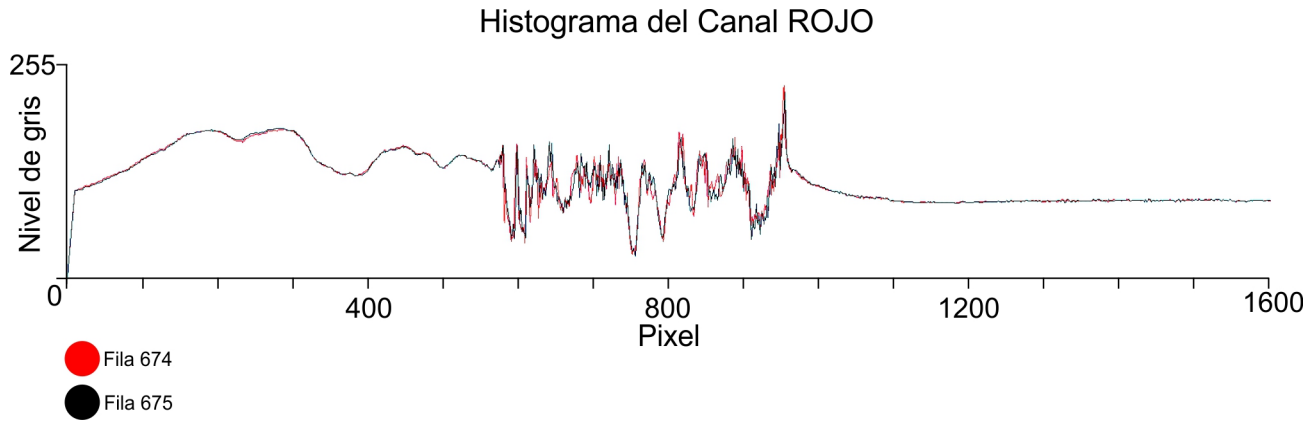


Figura 3.13: Histograma del canal rojo de las filas 674 y 675 de la Figura 3.10.

3.9. Histograma por columna de una imagen

El histograma por columna no es muy diferente al histograma por fila, al igual que el histograma anterior, estos brindan información del nivel de gris de la columna en la que se están tomando los pixeles. El eje X corresponde a la posición de los pixeles en la columna N y el eje Y el nivel de gris del pixel en particular.

3.10. Clustering

Los algoritmos de Machine Learning son una clase de algoritmos que permiten aprender a partir de datos y experiencias previas. En lugar de utilizar reglas programadas manualmente, los algoritmos de aprendizaje automático identifican patrones en los datos de entrada y generan modelos matemáticos que pueden utilizarse para realizar predicciones o tomar decisiones en nuevos datos.

Estos algoritmos se agrupan según los datos que analizan y hay dos tipos, aprendizaje supervisado y no supervisado. El aprendizaje supervisado consiste en analizar información asociada a etiquetas para predecir o clasificar información nueva y sirven para resolver problemas de regresión y clasificación. Mientras que el aprendizaje no supervisado consiste en analizar información sin etiqueta para encontrar alguna estructura en ella.

Entre las herramientas para la segmentación, existe el clustering, técnica no supervisada que se basa en identificar grupos de datos de tal manera que todos los datos del grupo (clúster) son datos con características similares mientras que los datos de los otros grupos son diferentes [Sanz, 2023].

La agrupación en clústeres también funciona como una herramienta independiente para obtener información sobre la distribución de datos o como un paso de pre procesamiento en otros algoritmos

[Singh, 2018].

A su vez, permite encontrar una relación oculta entre los elementos en el conjunto de datos, como en este caso, en la detección de distintos tipos de patrones en datos de imágenes.

3.11. K-Means Clustering

K-means es un algoritmo de agrupamiento por particiones que sirve separar un conjunto de datos en una cantidad conocida de grupos (clusters) utilizando la media y distancia euclidiana.

La media es el valor obtenido al sumar todos los elementos y dividirlo entre el total de elementos:

$$C = \frac{1}{m} \sum_{i=1}^m y_i$$

Donde m es la cantidad de elementos del grupo y y_i son las coordenadas de los elementos.

La distancia euclidiana es la más corta entre dos puntos del plano y su fórmula es:

$$d_E(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

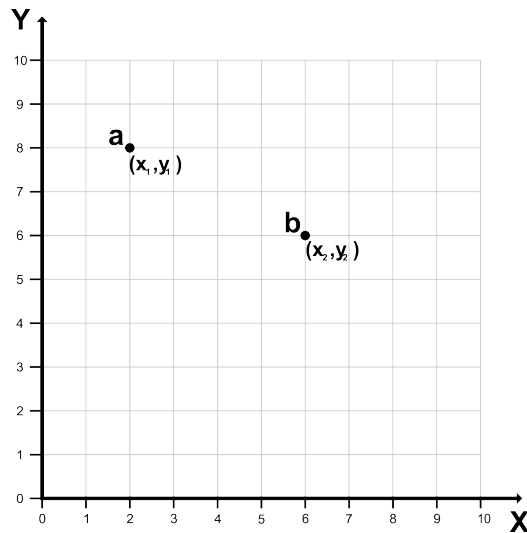


Figura 3.14: Puntos A y B.

Donde a es el punto (x_1, y_1) y b es el punto (x_2, y_2)

La Tabla 3.5 nos servirá como ejemplo.

Para obtener la media de los puntos de la Tabla 3.5, se suman las coordenadas x_1 y se dividen entre 3, después se suman las coordenadas x_2 y se dividen entre 3. Los centroides se forman a partir de las

Punto	x_1	x_2
a	1	3
b	2	2
c	3	1
MEDIA	2	2

Tabla 3.5: Media de los puntos A, B y C.

medias de los grupos por lo que a partir de ahora llamaremos centroides a las medias y clusters a los grupos.

Para encontrar el centroide más cercano se calcula la distancia entre los puntos y cada uno de los centroides.

$$C_{min} = \sum_{j=1}^n d(c_k, y_j)$$

Donde c_k representa a cada uno de los centroides y y_j representa a cada uno de los puntos del conjunto.

Por ejemplo, en el conjunto de la Figura 3.15

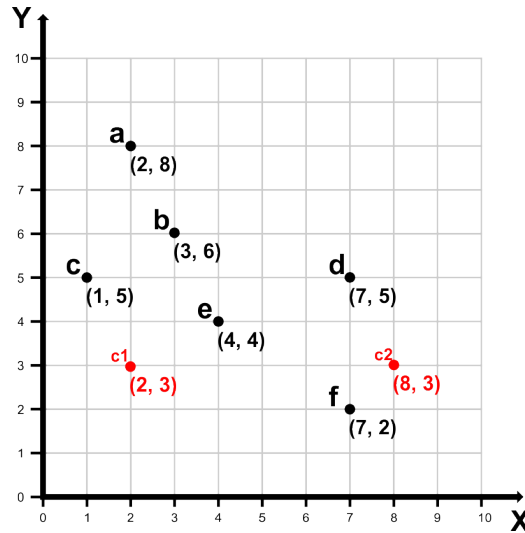


Figura 3.15: Conjunto de puntos.

Las distancias euclidianas a los centroides c_1 y c_2 están en la Tabla 3.6

Por lo que los datos se agrupan como se muestra en la Figura 3.16

La creación de los primeros centroides se puede realizar de manera aleatoria, teniendo en cuenta que los centroides deben estar suficientemente separados entre sí, para que los elementos no estén a distancias similares de todos los centroides [Colome Abril, 2012].

Punto	c_1	c_2
a	5	7.8
b	3.16	5.8
c	2.23	7.28
d	5.38	2.23
e	2.23	4.12
f	5.09	1.41

Tabla 3.6: Distancias euclidianas a los centroides c_1 y c_2 .

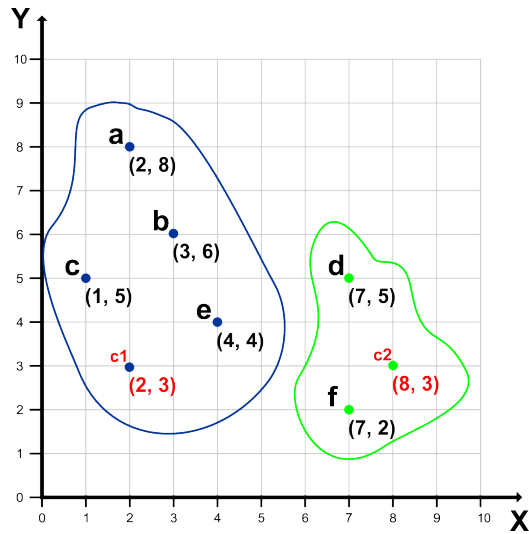


Figura 3.16: Grupos.

De esta manera, los pasos que sigue el algoritmo de k-means para encontrar los clusters en función de los datos son los siguientes:

- 1.- Asignar las k-medias, donde k es el número en el que se dividirán el conjunto.

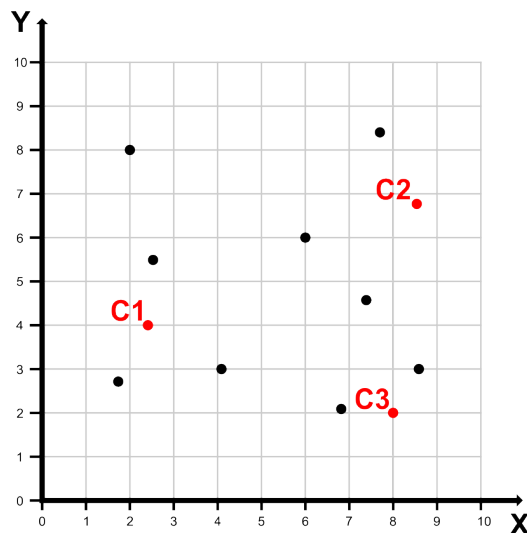


Figura 3.17: Paso 1 del algoritmo de k-means clustering.

2.-Asignar cada punto con la media (centroide) más cercana para formar los clusters (Utilizando la distancia euclidiana).

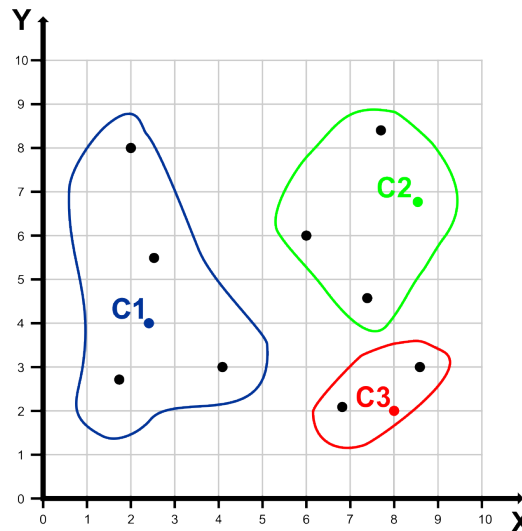


Figura 3.18: Paso 2 del algoritmo de k-means clustering.

3.-Recalcular las medias de los clusters.

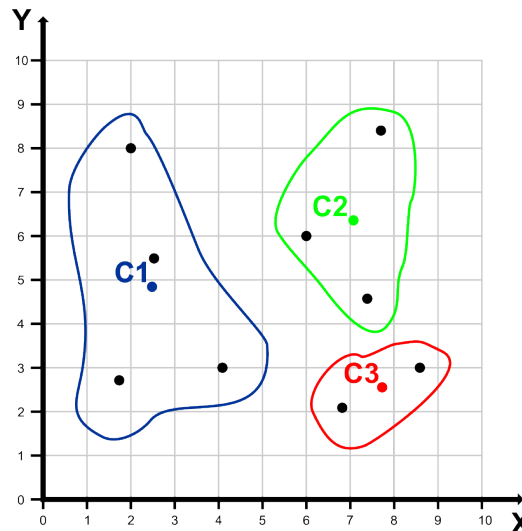


Figura 3.19: Paso 3 del algoritmo de k-means clustering.

4.- Se repiten los pasos 2 y 3 hasta que se cumpla un criterio de parada establecido por el usuario, los criterios de parada pueden ser:

- a) Las medias dejan de cambiar.
- b) Se alcanza un número máximo de iteraciones.

La cantidad de grupos en que se divida la información depende del problema y puede determinarse con un análisis previo. El algoritmo es sensible a la elección de medias iniciales por lo que no siempre

agrupará los elementos de la misma manera.

Página intencionalmente en blanco

Capítulo 4

Metodología

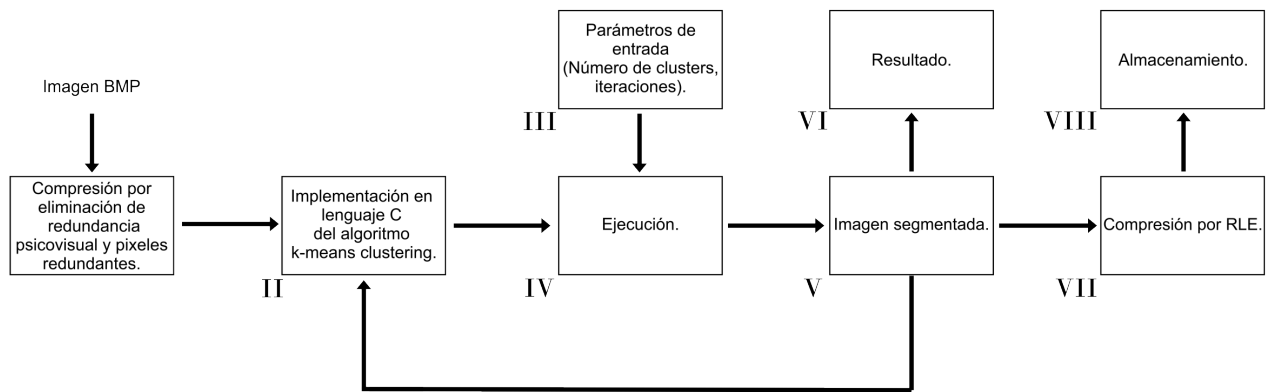


Figura 4.1: Diagrama de la metodología de este trabajo.

4.1. Compresión por eliminación de redundancia psicovisual y pixeles redundantes.

En esta sección, se hará la eliminación de las filas y columnas pares, sustentado en la similitud entre histogramas por filas y columnas con el objetivo de reducir el volumen de datos de la imagen.

Debido a lo anterior, para nuestro algoritmo la similitud entre líneas es prueba de que podemos eliminar una fila completa de la imagen. Esto lo hacemos para todas las filas pares. Con esto, nuestra imagen queda con una altura reducida a la mitad de su tamaño original.

La imagen de la Figura 3.10, tiene un tamaño de 1600 de ancho por 1200 pixeles de alto, al eliminar



Figura 4.2: Imagen de original sin filas pares de pixeles.

las filas pares, obtenemos la imagen de la Figura 4.2, con dimensiones de 1600 de ancho y 600 de alto. La eliminación de datos de la imagen es fácil de percibir, ya que, al eliminar las filas pares, da la percepción de estar “aplastada” y los objetos en ella pierden proporcionalidad.

De igual forma podemos eliminar columnas pares para reducir otro 50 % del tamaño de la imagen resultante sin filas, al hacerlo, reducimos el 50 % de sus datos, solo que esta vez en el ancho, dando como resultado la imagen de la Figura 4.3.



Figura 4.3: Imagen de la Figura 3.10 sin filas ni columnas pares.

La imagen de la Figura 4.3 tiene dimensiones de 800 pixeles de ancho por 600 de alto, al eliminar las filas y columnas pares, logramos una reducción del 75 % de los datos de la imagen original, reducción que, si bien se traduce a compresión con pérdida, esta pérdida no es significativa, conservó proporcionalidad

y no afectó la correcta representación de la información. Este método de compresión propuesto logró 2 tipos de compresión.

- Compresión por eliminación de píxeles redundantes: la similitud entre las líneas de píxeles no aporta información visual significativa.
- Compresión por eliminación de redundancia psicovisual: la eliminación de los datos de la imagen fue poco perceptibles y no comprometieron la calidad visual general de la imagen original.

Las imágenes de la Figura 4.4 muestran una comparativa real de proporcionalidad entre la imagen original y la imagen sin filas ni columnas pares.



Imagen original

Figura 4.4: Comparación de imágenes despues de la Fase I.

4.2. Implementación en lenguaje C del algoritmo k-means clustering para la segmentación de imágenes por color.

La segmentación es una operación esencial en el procesamiento digital de imágenes, ya que divide una imagen en sus partes constituyentes u objetos para separar lo que nos interesa del resto de la imagen [Palomino and Concha, 2009]. Esto nos facilita el análisis individual de los elementos y nos ayuda a tomar decisiones más informadas. Los atributos clave en la segmentación de imágenes incluyen

los componentes de color en imágenes, la luminancia en imágenes en blanco y negro, su textura, forma, entre otros.

Uno de los métodos utilizados para segmentar una imagen es la reducción de color mediante el uso del algoritmo de agrupamiento k-means clustering. En este enfoque, los niveles de gris de la imagen se agrupan en k grupos dentro del espacio de color RGB, lo que permite reducir la cantidad total de colores necesarios para representar la imagen.

Como se ha explicado en secciones anteriores, una imagen BMP está compuesta por píxeles, cada uno con su respectivo nivel de gris por canal.

Para llevar a cabo la segmentación de una imagen por color, se aplica el algoritmo de k-means clustering, donde los elementos del conjunto de datos corresponden a los niveles de gris de la imagen y los k grupos o clústeres representan los colores en los que se dividirá la imagen. El cálculo de los centroides y la asignación de los niveles de gris a los clústeres se realiza en el espacio de color RGB (Figura 3.7), ya que este contiene todos los elementos del conjunto de datos.

Dado que el algoritmo de k-means clustering puede requerir mucho tiempo de procesamiento para conjuntos de datos grandes, se utilizó la imagen de la jirafa (Figura 4.3) para las pruebas. Esta imagen se sometió a un preprocesamiento de la Fase I, donde se eliminaron filas y columnas, lo que resultó en una reducción del 75 % de los datos en comparación con la imagen original (Figura 3.10).

La imagen consta de 480,000 píxeles, pero en el programa implementado se procesan solo los niveles de gris sin repetir. El número de niveles de gris diferentes en la Figura 4.3 es de c, que es el total de datos del conjunto que buscamos reducir.

4.3. Parametros de entrada

Se realizaron pruebas con 1000, 100, 50, 10 y 3 clústeres, y la inicialización de los clústeres se realizó de forma aleatoria. En cada ejecución se midió el tiempo, y el criterio de parada se estableció cuando se alcanzaron un total de 100 iteraciones. El algoritmo se implementó en lenguaje C, se utilizó Gnuplot como herramienta para visualizar y comparar el movimiento de los clústeres en cada iteración, y las especificaciones del equipo de cómputo utilizado se detallan en la Tabla 4.1.

Sistema operativo	Windows 11 Home Single Language
RAM instalada	16.0 GB (15.4 GB usable)
Procesador	AMD Ryzen 5 5500U a 2.10 GHz

Tabla 4.1: Equipo utilizado para las pruebas.

Capítulo 5

Análisis y resultados

En este capítulo se muestran los resultados de implementar el algoritmo de k-means clustering a diferentes imágenes, se muestra gráficamente el movimiento de los clusters de cada iteración y se realiza el análisis de los resultados.

5.1. Jirafa segmentada con 3 clusters

La segmentación con 3 colores tomó 13:2 minutos para ejecutarse y como se ve en la Figura 5.1 en la primer iteración la distribución de los colores no fue favorable, cerca del 50 % de los datos pertenecían a un solo cluster y esto hizo que algunas partes del contorno de la jirafa se combinaran con el cielo. Lo interesante es que en las iteraciones siguientes la distribución de colores en cada cluster se fue equilibrando y esto permitió que existiera mayor contraste entre los 3 colores finales para la representación de la imagen y con ello cada uno de los elementos quedo definido y separado de los demás.

5.2. Jirafa segmentada con 10 clusters

Con 10 colores para la segmentación (Figura 5.2), el tiempo requerido fue de 14:43 minutos, 1 minuto mas que con 3 clusters. En la primera iteración, los colores no están bien distribuidos, lo que da la percepción de ver la misma imagen de 3 clústeres y 100 iteraciones. Sin embargo, a medida que avanzan las iteraciones, se agregan más detalles en la imagen; en el cielo, comienzan a aparecer algunas nubes y por primera vez, se aprecia el azul; en la jirafa, su color se acerca más al de la imagen original y las manchas presentan una mejor degradación de tonalidades; en los árboles, se nota que no todos tienen el mismo color y algunos troncos incluso presentan tonalidades cafés.

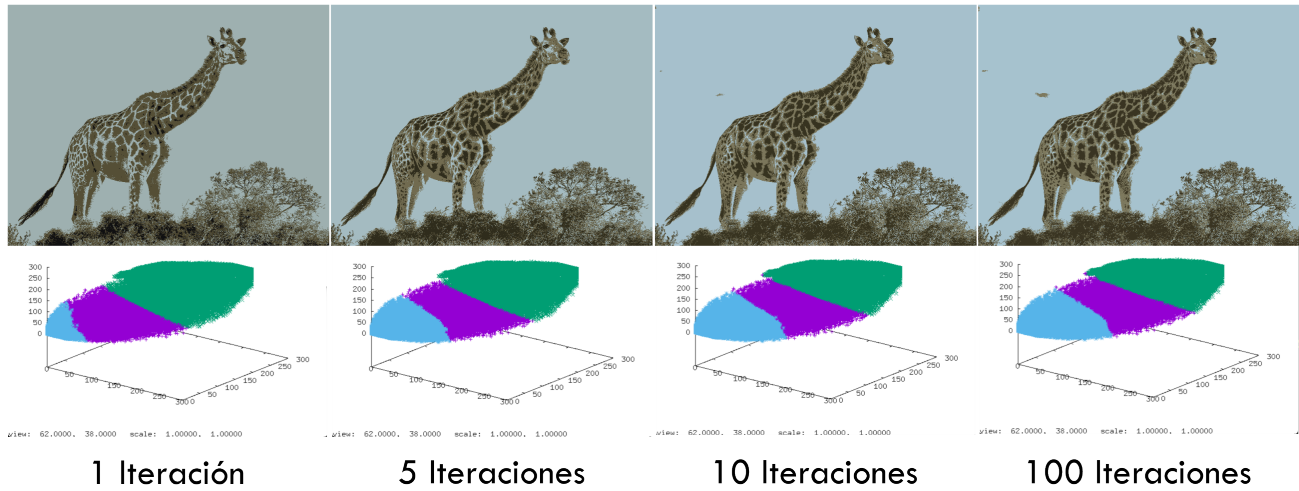


Figura 5.1: Jirafa segmentada con 3 clusters.

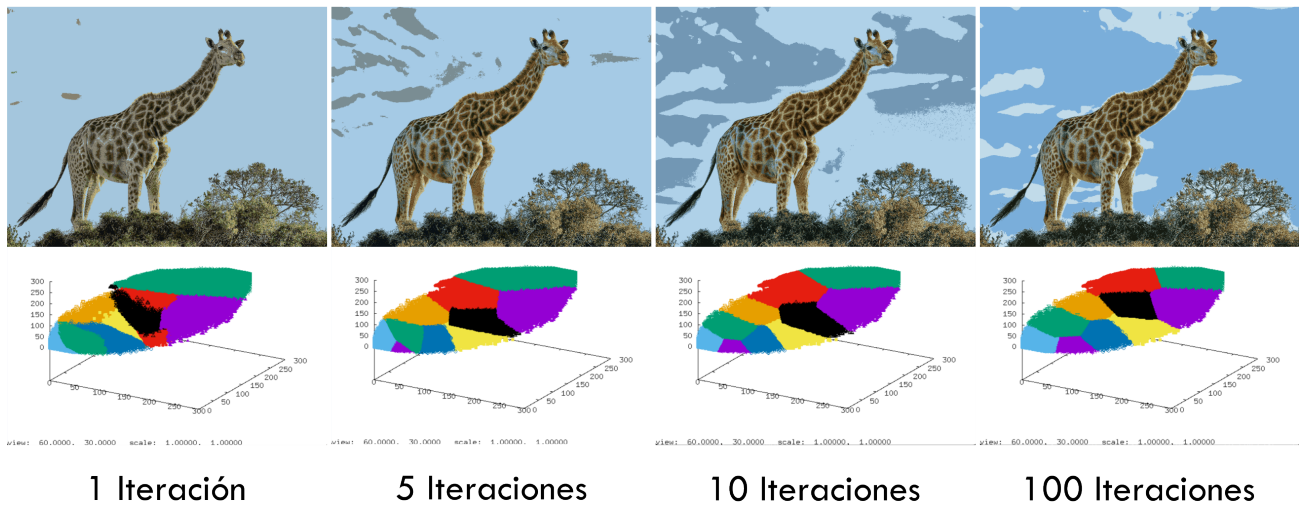


Figura 5.2: Jirafa segmentada con 10 clusters.

5.3. Jirafa segmentada con 50 clusters

Cuando la imagen se segmentó con 50 colores (Figura 5.3) la ejecución demoró 21:55 minutos, sin embargo, desde la primera iteración se pudo apreciar una mejora en la imagen en comparación a las anteriores (Figura 5.2), Aunque los colores aún no estaban bien distribuidos, a medida que avanzaban las iteraciones, se podían notar más detalles en los objetos de la imagen. El cielo toma distintas tonalidades

de azul y hay más nubes. Además, la jirafa ya no tenía manchas azules en la cara y los árboles adquirieron un tono verde más claro

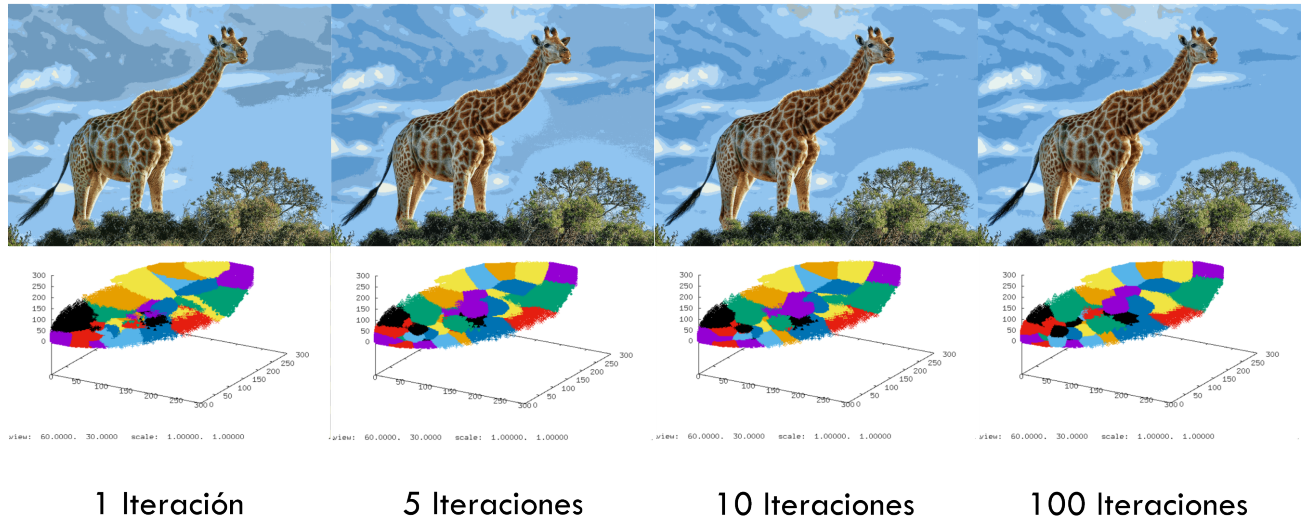


Figura 5.3: Jirafa segmentada con 50 clusters.

5.4. Jirafa segmentada con 100 clusters

Al llegar a la segmentación con 100 clusters (Figura 5.4), el tiempo requerido fue de 32:23 minutos. En la primera iteración, se observaron clústeres de diferentes tamaños, algunos más grandes que otros, sin embargo, a medida que avanzaron las iteraciones, los colores se distribuyeron de forma más uniforme. La imagen mostró más detalles en el cielo, evidenciando por primera vez la degradación en las nubes. En el caso de la jirafa y los árboles, resultó difícil notar algún cambio significativo en comparación con la segmentación anterior.

5.5. Jirafa segmentada con 1000 clusters

Al realizar la segmentación con 1000 clusters (Figura 5.5), el programa tardó 97:35 minutos en completarse. En la primera iteración, los clusters no estaban bien distribuidos. Sin embargo, la imagen presentó una gran calidad en la jirafa y los árboles, aunque en el cielo todavía se perdían algunos detalles. Con las iteraciones siguientes se comenzaron a formar las nubes y en la última iteración, finalmente obtuvimos una imagen que se asemeja bastante a la original, la jirafa estaba bien definida, los árboles

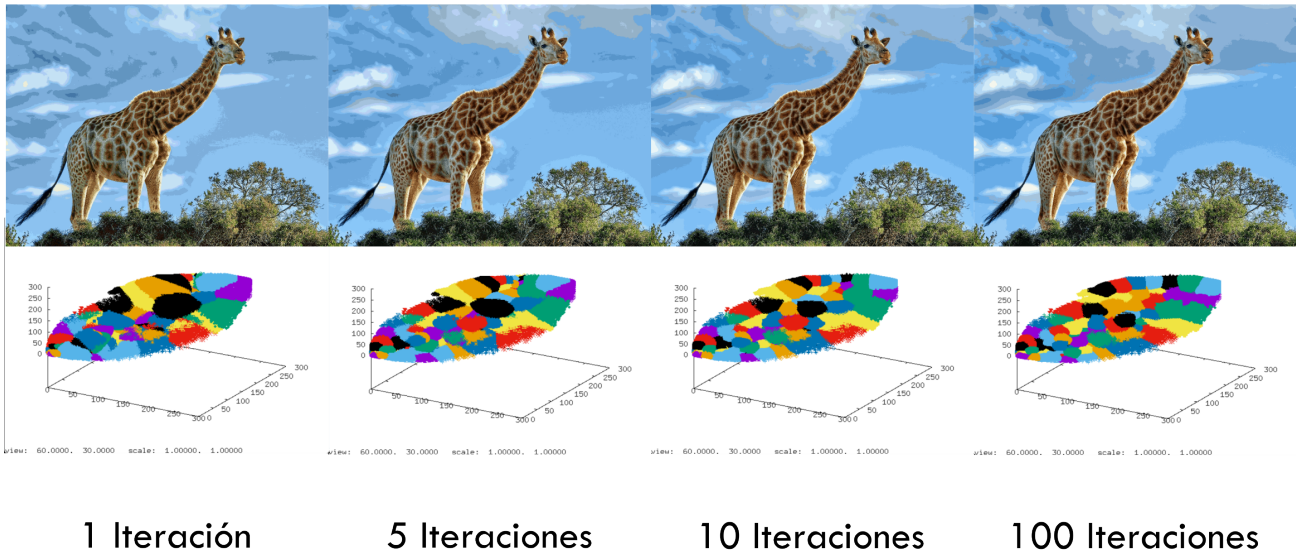


Figura 5.4: Jirafa segmentada con 100 clusters.

mostraban una variedad de colores y el cielo exhibía nubes con diferentes degradados de color, que se parecen mucho a la imagen original. Los clusters al final estaban bien distribuidos, lo que permitió que cada objeto de la imagen conservara la mayoría de los detalles que lo componían.

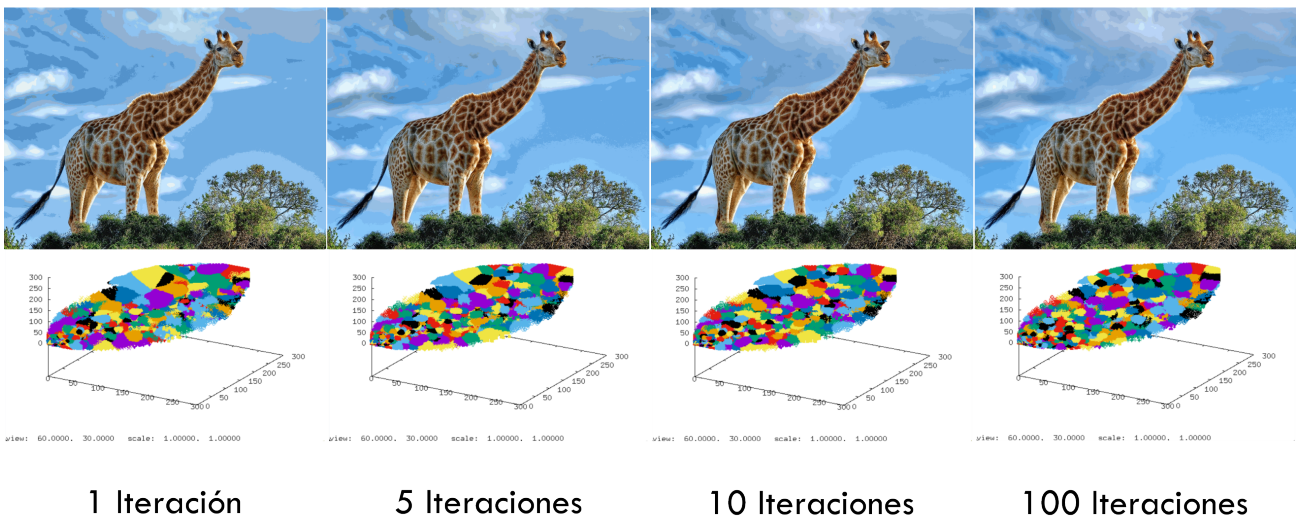


Figura 5.5: Jirafa segmentada con 1000 clusters.

5.6. Comparativa entre clusters

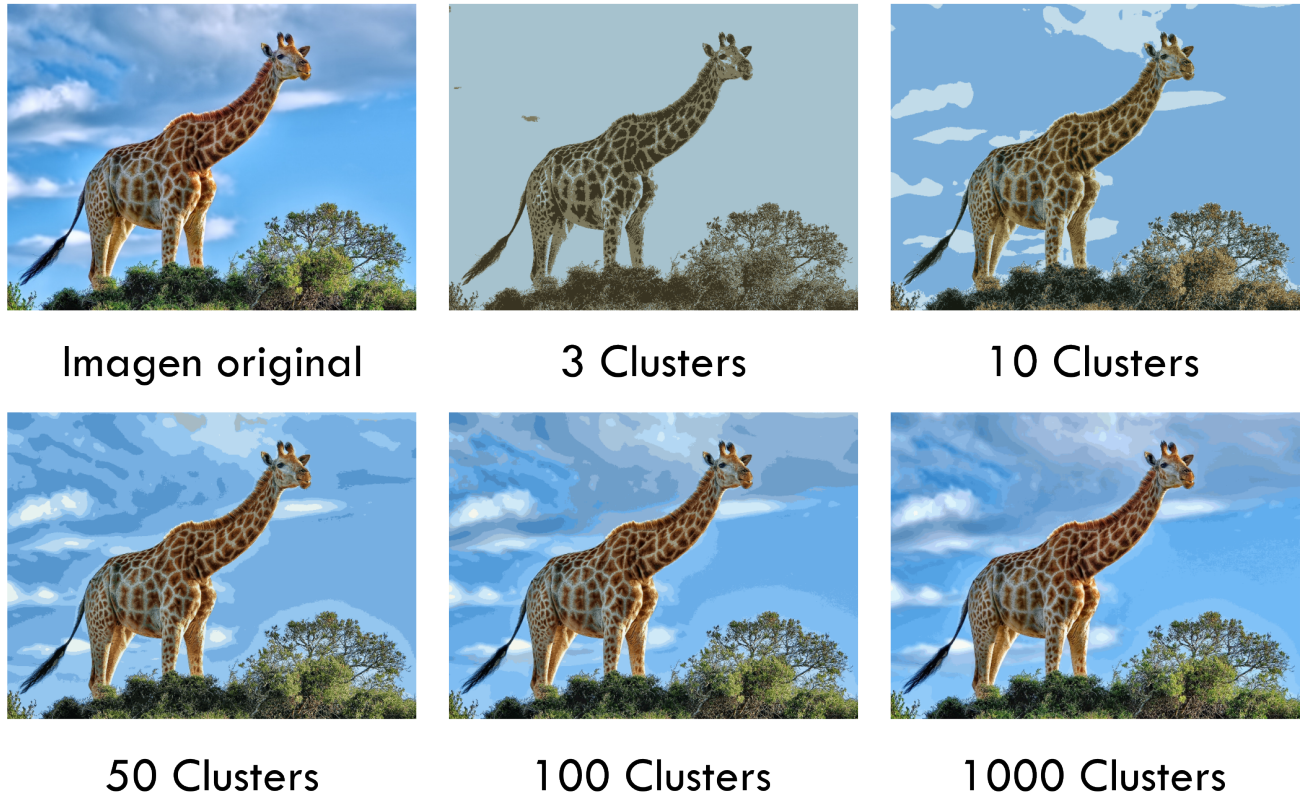


Figura 5.6: Comparación de los resultados con diferentes clusters.

Como se ve en las imágenes segmentadas de la Figura 5.6, sin importar la cantidad de clusters para segmentar la imagen, el método implementado se comporta correctamente, ya que destaca los objetos de mayor interés de la imagen sin que se superpongan. Esto es muy importante, pues la segmentación es un paso previo en métodos mas sofisticados para reconocer y etiquetar correctamente objetos de interés.

Cuando el número de clusters aumenta, la imagen resultante es más parecida a la original, solo que utiliza menos colores para representarla, en las prueba con 1000 clusters (Figura 5.5), se logró obtener una imagen similar a la original utilizando solo el .83 % de niveles de gris con los que contaba antes del método. Entre mas clusters se utilicen para segmentar la imagen, mayor será la similitud respecto a la imagen original.

5.7. Pruebas con diferentes imágenes

Durante las pruebas realizadas con imágenes adicionales, se pudo confirmar que los elementos importantes que componen las mismas se conservaban después de aplicar el algoritmo de segmentación por color.

Un ejemplo representativo es la imagen de Lena, como se puede observar en la Figura 5.7. Al realizar la segmentación con tan solo 10 clusters, se logró preservar los detalles de su rostro, el fondo y su sombrero. Esto permitió que todos los elementos de la imagen sean reconocibles y destaquen en relación a los demás.



Figura 5.7: Comparación de Lena con 10 clusters.

Durante el proceso de segmentación utilizando 10 clusters en las imágenes de los aguacates (Figura 5.8) y los platos de talavera (Figura 5.9), se observó un resultado similar. Todos los objetos importantes de las imágenes se mantuvieron, no se produjo una pérdida significativa de detalles y la principal diferencia entre las imágenes segmentadas y las originales radicó en las variaciones de tonalidades de color.

5.8. Almacenamiento de los resultados con RLE

El algoritmo de segmentación por color implementado demostró un buen funcionamiento, sin embargo, se observó que las imágenes segmentadas tienen un tamaño de archivo similar al de las imágenes originales. Con el fin de reducir el espacio de almacenamiento, se propone aplicar la compresión por RLE, aprovechando que las imágenes resultantes están compuestas por segmentos largos de niveles de

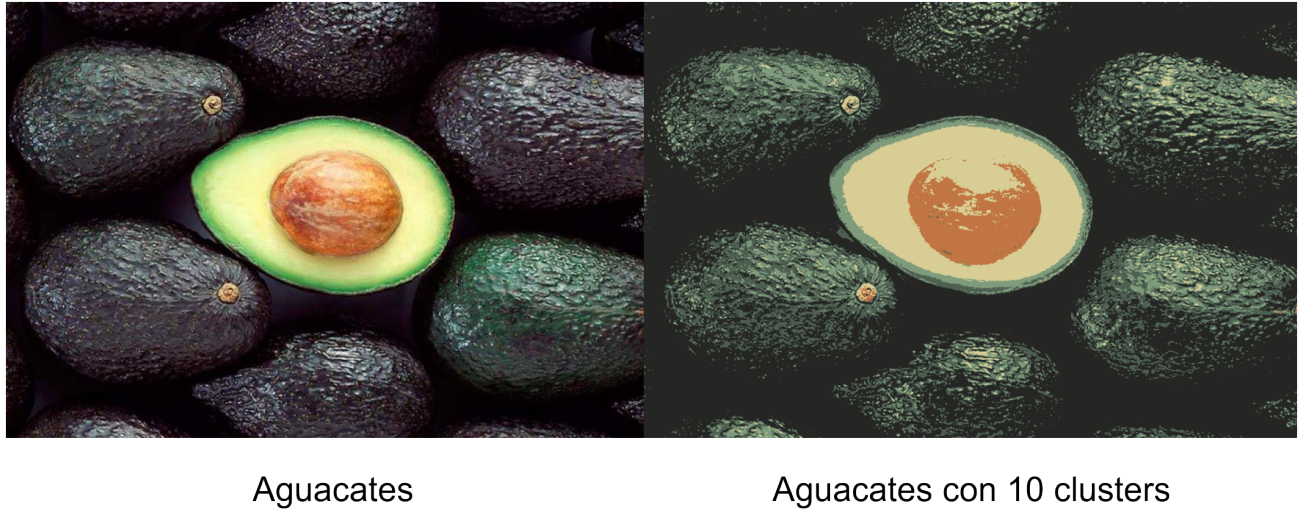


Figura 5.8: Comparación de aguacates con 10 clusters.

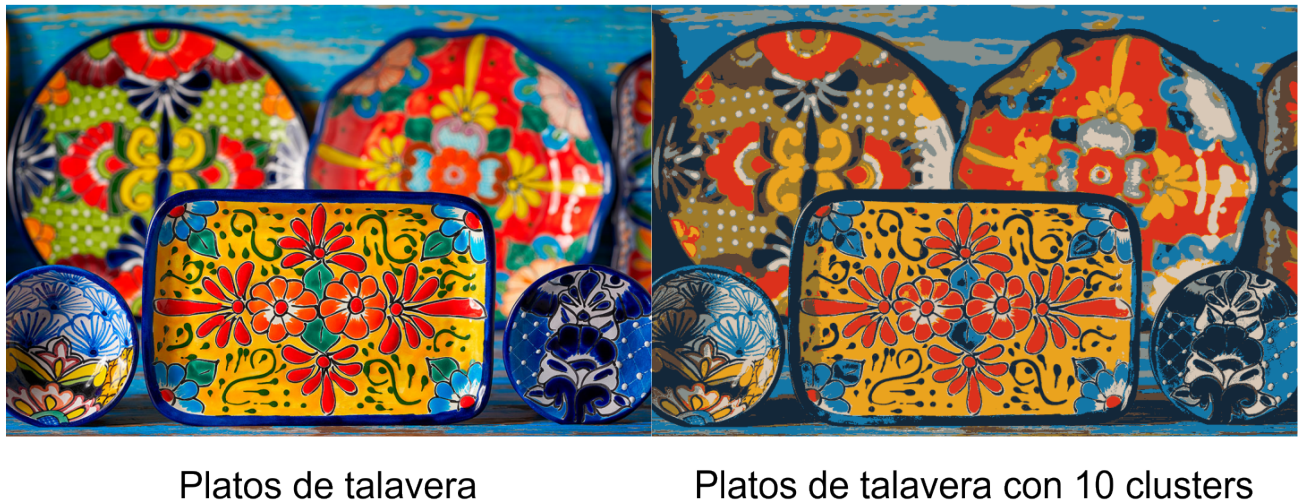


Figura 5.9: Comparación platos de talavera con 10 clusters.

gris. Esta técnica de compresión permitirá optimizar el almacenamiento de las imágenes segmentadas, aprovechando la redundancia de los datos presentes en ellas.

El peso de la imagen sin segmentar (Figura 4.3) es de 1,440,054 bytes, al aplicar RLE a la segmentación con 1000, 100, 50, 10 y 3 clusters obtenemos los valores de la Tabla 5.1.

En el caso de la segmentación con 1000 clusters, el tamaño de la imagen almacenada utilizando compresión por RLE fue mayor que el tamaño de la imagen original. Por lo tanto, se concluye que en este caso particular no es recomendable aplicar la compresión por RLE, ya que no resulta en una reducción del tamaño del archivo. Sin embargo, en el resto de las pruebas realizadas, se observó que la compresión por RLE logró reducir significativamente el tamaño de las imágenes, lo cual las hace más

IMAGEN	TAMAÑO EN BYTES
Imagen original (Figura 3.10)	5,760,054 bytes
Imagen sin filas ni culmnas pares (Figura 4.3)	1,440,054 bytes
Imagen con 1000 clusters (Figura 5.5)	1,466,374 bytes
Imagen con 100 clusters (Figura 5.4)	1,198,342 bytes
Imagen con 50 clusters (Figura 5.3)	1,064,960 bytes
Imagen con 10 clusters (Figura 5.2)	806,912 bytes
Imagen con 3 clusters (Figura 5.1)	477,262 bytes

Tabla 5.1: Tamaño en bytes después de comprimir con RLE

eficientes en cuanto a espacio en disco se refiere.

Conclusiones

Se ha desarrollado un método eficiente de segmentación de imágenes basado en el algoritmo de k-means clustering, iniciado por una etapa inicial de reducción de datos. La aplicación de esta técnica no solo ha mejorado el proceso de segmentación, sino que también ha conducido a una compresión de datos significativa al eliminar píxeles redundantes y redundancia psicovisual inherente en las imágenes.

La implementación eficiente del algoritmo de k-means en lenguaje C demostró su capacidad para lograr una segmentación precisa y rápida de imágenes.

Se implementó y evaluó técnicas de compresión de datos, destacando la eficacia de la compresión por eliminación de código redundante, especialmente mediante RLE, en la reducción del tamaño de imágenes segmentadas. Aunque se demostró su éxito en la mayoría de los casos, se identificó que en segmentaciones con gran cantidad de clusters, la eficacia de RLE podría ser limitada, lo que sugiere la exploración de enfoques más avanzados en investigaciones posteriores.

Página intencionalmente en blanco

Referencias

- [Colome Abril, 2012] Colome Abril, J. (2012). Aproximación al reajuste automático de centroides mediante la heurística de lloyd para resolver el problema de las k-medias.
- [Cruz Contreras et al., 2004] Cruz Contreras, A., González Robles, J. C., and Herrera Lozada, J. C. (2004). Procesamiento de imágenes: Estructura de archivos bmp. *Polibits*, 30:6–8.
- [Doe et al., 2019] Doe, J., Smith, A., and Johnson, B. (2019). A k-means clustering approach for color image segmentation in medical applications. *Journal of Medical Imaging*, 12(3):123–135.
- [González and Woods, 1999] González, R. C. and Woods, R. E. (1999). *Tratamiento digital de imágenes*. Addison-Wesley.
- [Lee et al., 2015] Lee, S., Kim, H., and Choi, S. (2015). Image grayscale histogram equalization and redundant pixel removal for image compression. *Journal of Multimedia*, 10(6):895–902.
- [Lezama, 2017] Lezama, J. (2017). Compresión de imágenes: Formato jpeg. *Revista de Educación Matemática*, 32(2):25–36.
- [Palomino and Concha, 2009] Palomino, N. L. S. and Concha, U. N. R. (2009). Técnicas de segmentación en procesamiento digital de imágenes. *Revista de investigación de Sistemas e Informática*, 6(2):9–16.
- [Rodríguez et al., 2017] Rodríguez, M., García, N., Pérez, L., and Martínez, E. (2017). Object detection in urban areas from satellite images using k-means clustering. *International Journal of Remote Sensing*, 38(10):2892–2916.
- [Salomón, 2007] Salomón, D. (2007). *Compresión de datos: la referencia completa*. Springer - Verlag London Limited.

- [Sanz, 2023] Sanz, F. (2023). Algoritmo k-means clustering: Aplicaciones y desventajas. <https://www.themachinelearners.com/k-means/>, 22 de Mayo de 2023.
- [Singh, 2018] Singh, S. (2018). Una introducción a la agrupación en clústeres <https://medium.datadriveninvestor.com/an-introduction-to-clustering-61f6930e3e0b>, 23 de Mayo de 2023.
- [Wang et al., 2015] Wang, J., Wu, P., and Hsu, C. (2015). Lossless image compression using row-column pixel redundancy elimination. *International Journal of Computer Theory and Engineering*, 7(2):121–125.

Código en Lenguaje C

.1. Programa de segmentación por k-means clustering

Este programa contiene la implementación del algoritmo de k-means Clustering para la segmentación de imágenes por color. Incluye el código fuente detallado con comentarios explicativos y se puede utilizar como referencia para comprender el proceso de segmentación y su aplicación en imágenes BMP. Los argumentos del programa son: La imagen a segmentar, nombre de la imagen segmentada, número de clusters y cantidad de iteraciones.

Código 1: Segmentación por k-means clustering.

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int NumClus, NIter, r,g,b, c, min, pos, i, it, j, k, cont=0, d, color, f, bandera=0,
    auxr=0, auxg=0, auxb=0; double dis[1010], step;
7 int S_STEP=11;
8 float minutos;
9 char cab[54];
10 long int tam, w, h;
11 time_t comienzo, final;
12 struct tm *tiempoComienzoPtr, *tiempoFinalPtr;
13
14 //Estructura de los pixeles
15 struct pix{
16     int b;
17     int g;
18     int r;
19     int color;
```

```

20  int cluster;
21  }*ar_pix, *ar_cluster;
22
23  //Funcion que toma los valores RGB de un pixel y los concatena
24  int colorpixel(int r, int g, int b) {
25      int color;
26      color=r*1000000+g*1000+b;
27      return color;
28  }
29
30  //Funcion que elimina los niveles de gris repetidos de la estructura de pixeles
31  void EliminaRepetidos() {
32      for(i=0; i<tam;i++)
33      {
34          for(j=i+1; j<tam;j++)
35          {
36              if (ar_pix[i].color == ar_pix[j].color)
37              {
38                  k=j;
39                  while (k<tam)
40                  {
41                      ar_pix[k]=ar_pix[k+1];
42                      ++k;
43                  }
44                  --tam;
45                  --j;
46              }
47          }
48      }
49  }
50  int main(int argc, char **argv){
51
52      FILE *img, *img2;
53      img = fopen(argv[1],"rb");
54      img2 = fopen(argv[2],"w");
55      NumClus = atoi(argv[3]);
56      NIter = atoi(argv[4]);
57
58      comienzo = time( NULL );
59      printf("Comienzo del programa: \n ");

```

```

60     system("date +%H:%M:%S");
61
62     //Llenado de la cabecera de la imagen segmentada
63     fread(cab,18,1,img);
64     fwrite(cab,18,1,img2);
65     fread(&w,4,1,img);
66     fwrite(&w,4,1,img2);
67     fread(&h,4,1,img);
68     fwrite(&h,4,1,img2);
69     fread(cab,28,1,img);
70     fwrite(cab,28,1,img2);
71     tam=w*h;
72
73     //Asignacion de memoria
74     ar_cluster=malloc(sizeof(struct pix)*NumClus);
75     ar_pix=malloc(sizeof(struct pix)*tam);
76
77     printf("Procesando la imagen\n");
78
79     //FOR que recorre la imagen y llena la estructura de pixeles
80     for(i=0; i<tam;i++)
81     {
82         ar_pix[i].b = fgetc(img);
83         ar_pix[i].g = fgetc(img);
84         ar_pix[i].r = fgetc(img);
85         ar_pix[i].color = colorpixel(ar_pix[i].r, ar_pix[i].g,ar_pix[i].b);
86     }
87
88     //Se eliminan los pixeles con nivel de gris repetido
89     EliminaRepetidos();
90     printf("La imagen tiene %ld colores \n ", tam);
91     printf("Creando los clusters\n");
92     srand(time(NULL));
93     step = (int)(rand() % S_STEP);
94
95     //Se crean los clusters
96     for(i=0; i<NumClus; i++)
97     {
98         ar_cluster[i]=ar_pix[(int)step*(1+i)];
99     }

```



```
100
101 //Inician las iteraciones
102 for ( it = 0; it < NIter; it++)
103 {
104     printf("Iteracion %d\n",it+1);
105     //Se calcula la distancia entre pixeles y clusters
106     for ( i = 0; i < tam; i++)
107     {
108         for ( c = 0; c < NumClus; c++)
109         {
110             dis[c] = sqrt((pow(ar_pix[i].r-ar_cluster[c].r,2))+(pow(ar_pix[i].b-
111                 ar_cluster[c].b,2))+(pow(ar_pix[i].g-ar_cluster[c].g,2)));
112         }
113         min=dis[0];
114         pos=0;
115         for(d=1 ; d<NumClus ; d++)
116         {
117             if (min>dis[d])
118             {
119                 min=dis[d];
120                 pos=d;
121             }
122         }
123         //Se asigna cada pixel al cluster mas cercano
124         ar_pix[i].cluster=pos;
125     }
126
127     //Se calculan las medias de los nuevos clusters
128     for (c = 0; c < NumClus; c++)
129     {
130         for ( i = 0; i < tam; i++)
131         {
132             if(ar_pix[i].cluster==c)
133             {
134                 auxr=auxr + ar_pix[i].r;
135                 auxg=auxg + ar_pix[i].g;
136                 auxb=auxb + ar_pix[i].b;
137                 ++cont;
138             }
139         }
140     }
```

```

139         if (cont != 0)
140         {
141             ar_cluster[c].r=auxr/cont;
142             ar_cluster[c].g=auxg/cont;
143             ar_cluster[c].b=auxb/cont;
144             ar_cluster[c].color=colorpixel(ar_cluster[c].r,ar_cluster[c].g,
                ar_cluster[c].b);
145             cont=0;
146             auxr=0;
147             auxg=0;
148             auxb=0;
149         }
150     }
151 }
152
153 printf("Imprimiendo imagen\n");
154 fseek(img,54,SEEK_SET);
155
156 //Se recorre la imagen original y se imprime la imagen segmentada
157 for ( i = 0; i < w*h; i++)
158 {
159     r=fgetc(img);
160     g=fgetc(img);
161     b=fgetc(img);
162     color=colorpixel(b,g,r);
163     for ( f = 0; f < tam; f++)
164     {
165         if (color==ar_pix[f].color)
166         {
167             fputc(ar_cluster[ar_pix[f].cluster].b,img2);
168             fputc(ar_cluster[ar_pix[f].cluster].g,img2);
169             fputc(ar_cluster[ar_pix[f].cluster].r,img2);
170             break;
171         }
172     }
173 }
174
175 printf("Listo\n");
176 final = time( NULL );
177 minutos=difftime(final, comienzo);

```

```
178     printf("Numero de segundos transcurridos desde el comienzo del programa: %f \n",
           minutos );
179     printf( "Fin del programa \n");
180     system("date +%H:%M:%S");
181 }
```

.2. Programa de compresión por RLE

Este programa presenta la implementación del algoritmo de compresión RLE. Permite comprimir imágenes segmentadas por k-means clustering eliminando la redundancia de datos. El código fuente está disponible en detalle, junto con instrucciones de uso y comentarios explicativos.

Código 2: Compresión por RLE.

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 //Estructura del pixel
6 struct pix{
7     int b;
8     int g;
9     int r;
10    int color;
11    int cluster;
12 }*ar_pix;
13
14 //Estructura de los datos para la escritura del .bin
15 struct datos
16 {
17     long int color;
18     short int contador;
19 }cadenaRLE;
20
21 //Funcion que toma los valores RGB de un pixel y los concatena
22 int colorpixel(int r, int g, int b) {
23     int color;
24     color=r*1000000+g*1000+b;
25     return color;
26 }
```

```

27
28 int main(int argc, char **argv){
29     char cab[54], cab2[54];
30     int tam, w, h, i, aux=0, cont=1;
31     FILE *img, *arch;
32     img = fopen(argv[1], "rb");
33     arch = fopen(argv[2], "wb");
34     fread(cab2, 54, 1, img);
35     rewind(img);
36     fread(cab, 18, 1, img);
37     fread(&w, 4, 1, img);
38     fread(&h, 4, 1, img);
39     fread(cab, 28, 1, img);
40     tam=w*h;
41     ar_pix=malloc(sizeof(struct pix)*tam);
42
43     //Ciclo FOR para recorrer la imagen y llenar la estructura AR_PIX
44     for(i=0; i<tam; i++)
45     {
46         ar_pix[i].b = fgetc(img);
47         ar_pix[i].g = fgetc(img);
48         ar_pix[i].r = fgetc(img);
49         ar_pix[i].color = colorpixel(ar_pix[i].r, ar_pix[i].g, ar_pix[i].b);
50     }
51     fwrite(cab2, 54, 1, arch);
52
53     //Llenado del .bin con la imagen comprimida
54     while (aux<tam)
55     {
56         if (ar_pix[aux].color==ar_pix[aux+1].color)
57         {
58             aux++;
59             cont++;
60         }
61         else
62         {
63             cadenaRLE.color=ar_pix[aux].color;
64             cadenaRLE.contador=cont;
65             fwrite(&cadenaRLE, sizeof(struct datos), 1, arch);
66             aux++;

```

```
67         cont=1;
68     }
69
70 }
71 cadenaRLE.color=10;
72 cadenaRLE.contador=-1;
73 fwrite(&cadenaRLE, sizeof(struct datos),1,arch);
74 }
```

.3. Programa de descompresión de archivos RLE

Este programa proporciona la implementación del algoritmo de descompresión para archivos comprimidos en formato RLE. Permite revertir el proceso de compresión y restaurar la imagen original segmentada. El código fuente se encuentra detallado y comentado para facilitar su comprensión y uso.

Código 3: Recuperación RLE.

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 //Estructura de datos para el llenado del .bin
6 struct datos
7 {
8     long int color;
9     short int contador;
10 }cadenaRLE;
11
12 //Funcion que recupera el valor ROJO del entero COLOR
13 int red(int color) {
14     int r;
15     r=color/1000000;
16     return r;
17 }
18
19 //Funcion que recupera el valor VERDE del entero COLOR
20 int green(int color, int r) {
21     int g;
22     g=(color-(r*1000000))/1000;
23     return g;
```

```
24 }
25
26 //Funcion que recupera el valor AZUL del entero COLOR
27 int blue(int color, int r, int g) {
28     int b;
29     b=color-(r*1000000)-(g*1000);
30     return b;
31 }
32
33 int main(int argc, char **argv){
34     FILE *img, *arch;
35     char cab[54];
36     int i,r, g, b;
37     arch = fopen(argv[1],"rb");
38     img = fopen(argv[2],"wb");
39     fread(cab,54,1,arch);
40     fwrite(cab,54,1,img);
41
42     //WHILE que recorre el .bin y escribe la imagen
43     while (cadenaRLE.contador != -1)
44     {
45         fread(&cadenaRLE, sizeof(struct datos),1,arch);
46         r=red(cadenaRLE.color);
47         g=green(cadenaRLE.color,r);
48         b=blue(cadenaRLE.color,r,g);
49         for ( i = 0; i < cadenaRLE.contador; i++)
50         {
51             fputc(b,img);
52             fputc(g,img);
53             fputc(r,img);
54         }
55     }
56 }
57
58 }
```