**Algoritmos para la Compresión de imágenes**

|  |  |  |  |
| --- | --- | --- | --- |
| Julian David Andrade Gomez  EAFIT  Colombia  Jdandradeg@eafit.edu.co | Martin Diaz  EAFIT  Colombia  Mdiazv1@eafit.edu.co | Simón Marín Universidad Eafit Colombia smaring1@eafit.edu.co | Mauricio Toro  Universidad Eafit  Colombia  mtorobe@eafit.edu.co |

# **RESUMEN**

Hay diferentes softwares que pueden reconocer si un ganado está enfermo mediante fotos de este, en el siguiente trabajo se quiere lograr comprimir imágenes de grandes bases de datos de fotos de ganado, pero manteniendo una alta eficiencia de reconocimiento de las enfermedades, Este problema es importante ya que el uso de inteligencia artificial se usa más cada día, y reconocer algoritmos de compresión de imágenes que no interferirán con la eficiencia de la inteligencia artificial puede llegar a ser sumamente importante, no solo para este problema en específico, sino también para otros como los que se mencionaran a continuación: El uso de inteligencia artificial para la toma de decisiones en el mantenimiento de ganado, alimentación y uso de recursos en la ganadería. El uso de drones y inteligencia artificial para la localización de vacas u otros animales. El uso de compresión de imágenes para optimizar los videojuegos.

## **Palabras clave**

|  |
| --- |
| Algoritmos de compresión, aprendizaje de máquina,  aprendizaje profundo, ganadería de precisión, salud animal. |

# **1. INTRODUCCIÓN**

La razón por la cual se realizo este trabajo fue para lograr gastar menos memoria a la hora de tener una base de datos de fotos de ganado, las cuales se usarán para clasificar su salud.

# **Problema**

El peso en memoria de una imagen presenta varias problemáticas a la hora de guardar una alta cantidad de imágenes o al buscar una alta velocidad para enviar imágenes por internet, es debido a esto que existen diversos métodos de comprensión de imágenes que permite que están se almacenen en una cantidad mucho inferior de memoria.

**1.2 Solución**

En este trabajo, utilizamos una red neuronal convolucional para clasificar la salud animal, en el ganado vacuno, en el contexto de la ganadería de precisión (GdP). Un problema común en la GdP es que la infraestructura de la red es muy limitada, por lo que se requiere la compresión de los datos. Para esta comprensión desarrollamos un algoritmo de comprensión sin perdidas que reduce la cantidad de colores de la imagen pero manteniendo la cantidad de pixeles, ya que pensamos que los pixeles tendrán una mayor importancia para el reconocimiento de la imagen, mientras que le uso de muchos colores no necesariamente aporte tanto a esto mismo. El algoritmo con perdidas que se desarrollo es llamado k-means clustering.

**1.3 Estructura del artículo**

En lo que sigue, en la Sección 2, presentamos trabajos relacionales con el problema. Más adelante, en la Sección 3, presentamos los conjuntos de datos y los métodos utilizados en esta investigación. En la Sección 4, presentamos el diseño del algoritmo. Después, en la Sección 5, presentamos los resultados. Finalmente, en la Sección 6, discutimos los resultados y proponemos algunas direcciones de trabajo futuras.

**2. TRABAJOS RELACIONADOS**

## En lo que sigue, explicamos cuatro trabajos relacionados. en el dominio de la clasificación de la salud animal y la compresión de datos. en el contexto del PLF.

## **2.1 Análisis de algoritmos de compresión de imágenes para su implementación en dispositivos móviles**

El autor de este trabajo se propuso investigar sobre diversos algoritmos de compresión con perdida, con el fin de desarrollar una aplicación móvil que permitiese aplicarles esta compresión a las fotos almacenadas. Su objetivo era reducir la cantidad de memoria de las imágenes, manteniendo una alta calidad de estas (que la perdida se minimizara).

El autor concluye que el algoritmo más eficiente para una aplicación móvil sería el de la transformación discreta de cosenos, ya que tiene una complejidad de , la cual es suficientemente rápida y la perdida de la imagen es muy baja.

[1]

## **2.2 Análisis de la compresión de imágenes utilizando clustering bajo el enfoque de colonia de hormigas**

La autora de este trabajo nos habla del clustering, un algoritmo que trata de agrupar pixeles similares en grupos. Este documento propone y compara los diferentes métodos que usan el clustering, con un método especifico de clustering llamado “ant based clustering”, el cual se basa en el comportamiento de las hormigas a la hora de recoger alimento, (estas dejan un camino de feromonas para lograr llevarse todo el alimento de un lugar), bajo esta idea se desarrolla un algoritmo de clustering.

Dibujo en blanco y negro

Descripción generada automáticamente con confianza media

[3]

## **2.3 Compresión fractal de imágenes**

En este trabajo, se nos habla de como los patrones fractales se pueden usar para reconstruir una imagen, y como teóricamente se puede seguir reduciendo la perdida de la imagen a medida que se sigue aplicando el algoritmo. En el paper se nos habla de cómo esta solución se conocía desde antes, pero era muy lenta, pero mediante la ayuda de algoritmos genéticos.

Forma, Flecha

Descripción generada automáticamente

[4]

## **2.4 Algoritmos de compresión de imágenes de alta resolución sin pérdidas.**

El autor habla de algoritmos de compresión sin perdida (es decir, la imagen original se puede volver a conseguir tras la compresion). Su algoritmo sin embargo necesita que las imágenes contengan pocos colores para su funcionamiento, por lo que su algoritmo sirve mas que todo para imágenes satelitales o mapas topológicos.

Interfaz de usuario gráfica, Mapa

Descripción generada automáticamente

[2]

## **3. MATERIALES Y MÉTODOS**

En esta sección, explicamos cómo se recogieron y procesaron los datos y, después, diferentes alternativas de algoritmos de compresión de imágenes para mejorar la clasificación de la salud animal.

## **3.1 Recopilación y procesamiento de datos**

Recogimos datos de *Google Images* y *Bing Images* divididos en dos grupos: ganado sano y ganado enfermo. Para el ganado sano, la cadena de búsqueda era "cow". Para el ganado enfermo, la cadena de búsqueda era "cow + sick".

En el siguiente paso, ambos grupos de imágenes fueron transformadas a escala de grises usando Python OpenCV y fueron transformadas en archivos de valores separados por comas (en inglés, CSV). Los conjuntos de datos estaban equilibrados.

El conjunto de datos se dividió en un 70% para entrenamiento y un 30% para pruebas. Los conjuntos de datos están disponibles en https://github.com/mauriciotoro/ST0245-Eafit/tree/master/proyecto/datasets .

Por último, utilizando el conjunto de datos de entrenamiento, entrenamos una red neuronal convolucional para la clasificación binaria de imágenes utilizando *Teachable Machine* de Google disponible en https://teachablemachine.withgoogle.com/train/image.

## **3.2 Alternativas de compresión de imágenes con pérdida**

## En lo que sigue, presentamos diferentes algoritmos usados para comprimir imágenes con pérdida.

**3.2.1 Tallado de costuras**

Este algoritmo es un algoritmo de cambio de tamaño de imágenes, principalmente se usa para adaptar una imagen a diferentes tamaños de pantallas, pero también se puede usar para comprimir una imagen en sí. El algoritmo se basa en darle una importancia a cada píxel y luego buscar un camino de arriba debajo de pixeles que la suma de importancia de estos sea la mínima, luego se borran estos pixeles.

Un dibujo de una persona

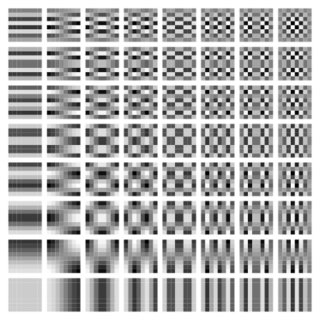
Descripción generada automáticamente con confianza baja

En la imagen los caminos rojos representan los caminos que serán borrados.

La complejidad depende del método que se use para definir la importancia de los pixeles, pero en los más simples se logra una complejidad de donde n es los pixeles horizontales, m los verticales y k la cantidad de columnas de pixeles que se desean borrar.

**3.2.2 transformación de coseno discreto**

**E**ste algoritmo se basa en describir una matriz de 8x8 pixeles como suma de 64 funciones de coseno (funciones con diferentes frecuencias), luego se ignora la información de las funciones de coseno con alta frecuencia, pues estas no le aportan mucho cambio a la imagen, lo que logra comprimir la cantidad de información que se debe tener.



La imagen muestra las diferentes frecuencias de funciones coseno que se usan para representar cualquier imagen de 8x8.

Este algoritmo se debe llevar a cabo en imágenes de , en caso de que sea rectangular, se puede expresar como la unión de varios cuadrados. La complejidad del algoritmo para una imagen de pixeles es .

**3.2.3 Compresión fractal**

La compresión fractal tata de encontrar las similitudes en una imagen, por lo que este lo que trata de hacer es tener unas imágenes base y guardar las traslaciones y rotaciones que se les aplica a estas para llegar a la posición donde deberían estar en la imagen original, así en vez de guardar muchos pixeles, se guardan unos pocos y las transformaciones que se les debe aplicar a estos para llegar a todos los puntos de la imagen donde deben estar.

Una mujer con un sombrero

Descripción generada automáticamente con confianza media

Los algoritmos mas eficientes logran comprimir las imágenes en y descomprimirlas en

**3.2.4 Escalado de imágenes**

## Este algoritmo consiste en tomar una imagen y aumentar o reducir su cantidad de pixeles (reducir la cantidad disminuye la calidad visual), los algoritmos más simples simplemente multiplican la posición del pixel por un escalar y para la nueva imagen se toma el color del pixel más cercano de los originales al multiplicarse por este escalar, este proceso se puede lograr en .

Imagen que contiene computadora, tabla, iluminado

Descripción generada automáticamente

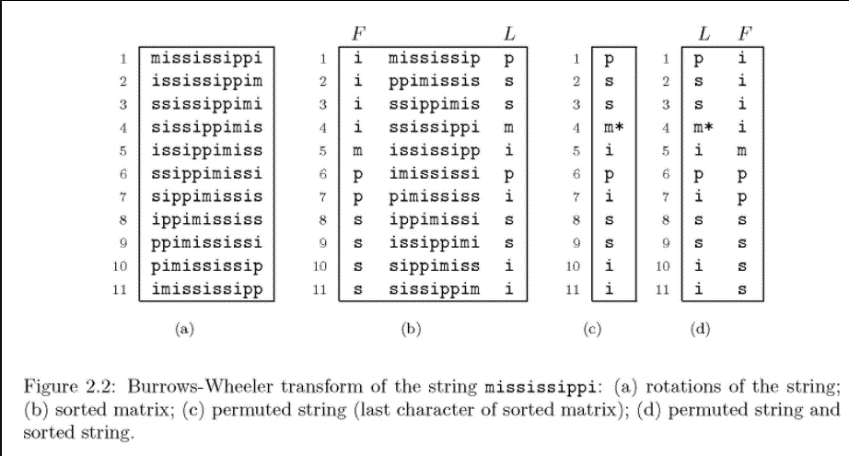
En la imagen se puede ver a la derecha la imagen original y a la izquierda una que se le redujo la cantidad de pixeles y luego se le volvió a aumentar.

**3.3 Alternativas de compresión de imágenes sin pérdida**

## En lo que sigue, presentamos diferentes algoritmos usados para comprimir imágenes sin pérdida.

## **3.3.1 Compresión Transformada de Burrows-Wheeler** (BWT)

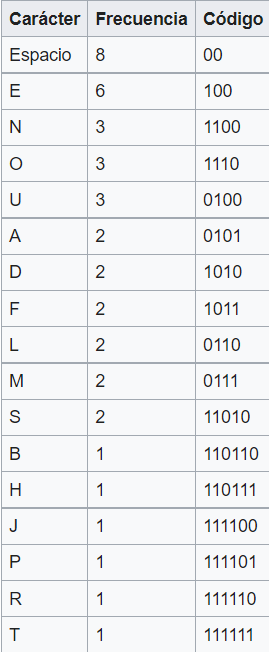
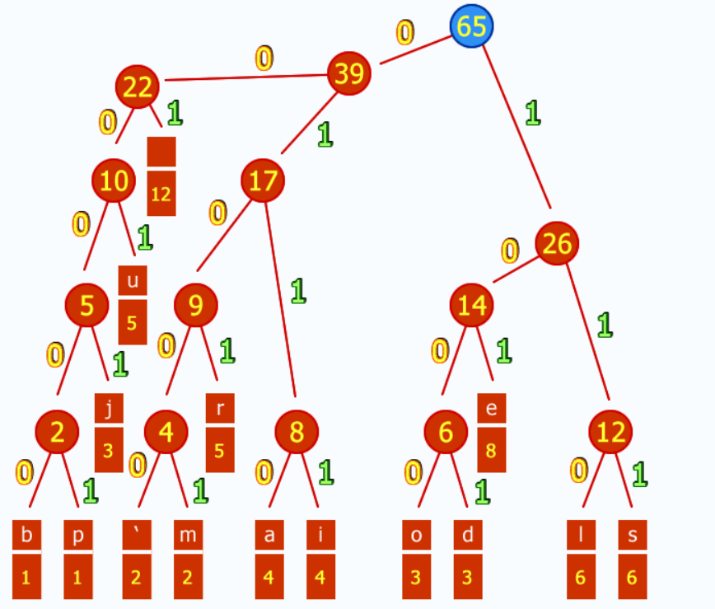
## Es un algoritmo que toca bloques de datos, como cadenas, y los reorganiza en ejecuciones de caracteres similares. Despues de la transformación, el bloque de salida contiene exactamente los mismos elementos de datos antes de que comenzara, pero en diferente orden.



# **3.3.2 Codificación de Huffman**

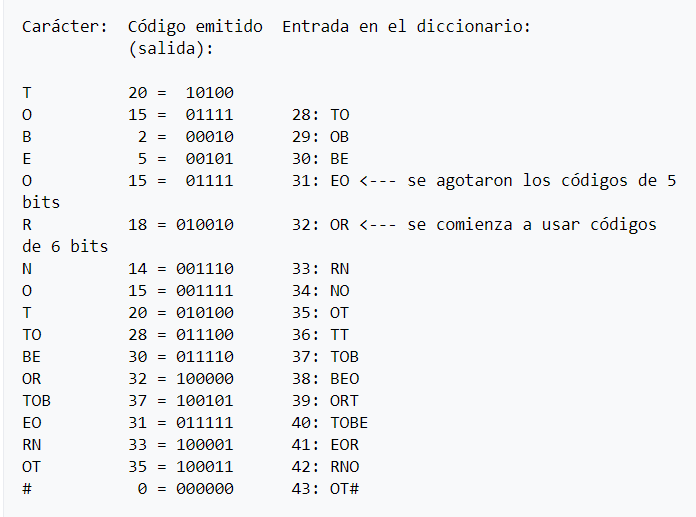
Este algoritmo crea una estructura arbórea ordenada con todos los símbolos y frecuencia con que aparecen. Las ramas se construyen en forma recursiva comenzando con los símbolos menos frecuentes.





**3.3.3 Algoritmo de Lempel-Ziv-Welch (LZW)**

Este algoritmo es una forma de compresión sustitutivo. En esta forma de compresión, una cadena de caracteres específica, única se sustituye por una referencia a esa frase, que se mantiene en un diccionario. Los datos resultantes se comprimen porque la referencia a la frase repetida es mucho menor. Supongamos que fueron comprimiendo la frase, "el tiburón del mako es la raza más rápida del tiburón". Porque la palabra "el" se repite, puede ser reemplazado por un marcador de posición como "\*".

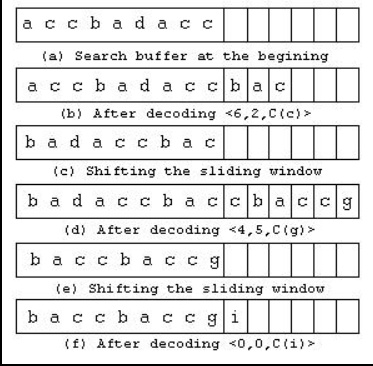


**3.3.4 Algoritmos de Lempel-Ziv 1977 (LZ77)**

En este algoritmo el codificador analiza el texto como una secuencia de caracteres, mediante una ventana deslizable compuesta por dos partes; un [buffer](https://www.ecured.cu/Buffer) de anticipación que es la opción que está a punto de ser codificada y un [buffer](https://www.ecured.cu/Buffer) de búsqueda (la ventana en sí), que es la parte dónde se buscan secuencias iguales a las existentes en el [buffer](https://www.ecured.cu/Buffer) de anticipación. Para codificar el contenido, o parte de él, del [buffer](https://www.ecured.cu/Buffer) de anticipación, se busca la secuencia igual en el [buffer](https://www.ecured.cu/Buffer) de búsqueda y la codificación resulta en indicar esta repetición como una tripleta **[offset, longitud, carácter siguiente].**

Donde:

* Offset es la distancia desde el principio del [buffer](https://www.ecured.cu/Buffer) de anticipación hasta el comienzo de la secuencia repetida.
* Longitud es la cantidad de caracteres repetidos.
* Carácter siguiente es el símbolo siguiente a la secuencia en el [buffer](https://www.ecured.cu/Buffer) de anticipación



## **4. DISEÑO E IMPLEMENTACIÓN DE LOS ALGORITMOS**

## En lo que sigue, explicamos las estructuras de datos y los algoritmos utilizados en este trabajo. Las implementaciones de las estructuras de datos y los algoritmos están disponibles en Github[[1]](#footnote-2).

## **4.1 Estructuras de datos**

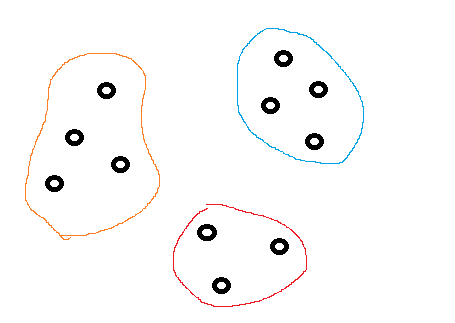
Para el algoritmo de compresión con perdidas no se necesita ninguna estructura de datos compleja, ya que con los arrays de Python y numpy es suficiente.

**4.2 Algoritmos**

En este trabajo, proponemos un algoritmo de compresión que es una combinación de un algoritmo de compresión de imágenes con pérdidas y un algoritmo de compresión de imágenes sin pérdidas. También explicamos cómo funciona la descompresión para el algoritmo propuesto.

**4.2.1 Algoritmo de compresión de imágenes con pérdida**

El algoritmo con pérdidas que se utilizará será k-menas clustering, este algoritmo clasificado como un algoritmo de inteligencia artificial sin supervisión, se usa para separar un gran grupo de datos en k grupos q cumplen con cierta similitud, como se puede observar en la siguiente imagen:



En la imagen anterior se ve como el algoritmo puede separar diferentes puntos en el planto en tres grupos diferentes por su cercanía.

en el caso de la compresión de imágenes este algoritmo se usa para hacer una reducción de la cantidad de colores. En el caso especifico de este trabajo, cada píxel tiene un valor de 0 a 255 para determinar su color, el algoritmo de k-means nos permitirá reducir la cantidad de colores a solo 16, es decir que cada pixel solo tendrá un valor de 0-16, reduciendo la cantidad necesaria de byts para guardar cada pixel.

El algoritmo entonces clasificara la cercanía de dos pixeles por su valor de color, mientras mas cercanos estén, mas posible es que queden en el mismo grupo, y finalmente todos los pixeles en el mismo grupo se les determinara un único valor de píxel (se forman 16 grupos).

El algoritmo realiza los siguientes pasos:

1. El algoritmo genera k centroides aleatoriamente (en este caso k=16 y los centroides son valores de color de 0-255 aleatoriamente).
2. Para cada píxel de la imagen, encuentra cual de los centroides es mas cercano a él, es decir, que la diferencia de valor de color entre el centroide y el pixel es mínima.
3. Recalcula los centroides. Esto lo hace tomando el valor promedio de color de todos los pixeles que pertenecen a cada centroide.
4. Se repite el paso 2 y 3 hasta que al recalcular los centroides estos no cambien.

Al final cada píxel se cambia por el color de su centroide.

En la siguiente imagen podemos ver el resultado de esta compresión:

**Imagen original**

**Imagen tras la compresión**

**4.2.2Algoritmo de compresión de imágenes sin pérdida**

Para este caso usamos el algoritmo LZ77, en cual es un algoritmo basado en diccionario que codifica cadenas largas en tokens cortos y reemplaza frases en el diccionario con tokens más pequeños para lograr la compresión. Busca las coincidencias para generar una triple marca, donde la primera es el número de deposiciones que necesitaríamos mover hacia atrás para encontrar el inicio de la cadena que coincide, la segunda representa la longitud después del encuentro, y la tercer representa el carácter que se encuentra después de la cadena que coincide. Al final de la descompresión se debe tener otra vez la cadena de pixeles exactamente igual la inicial, para esto se toma cada tripleta, primero se busca en el buffer y luego se escribe en la cadena final.

### Secuencia básica: Compresor

* Guardar el tamaño del archivo a comprimir.
* Repetir hasta que no haya más *bytes* para comprimir.
* Escanear el [buffer](https://es.wikipedia.org/wiki/Buffer_de_datos) de entrada comenzando en **posición\_actual - tamaño\_de\_ventana\_corrediza** hasta el *byte* actual que estamos comparando. (Notar que el descompresor no puede copiar *bytes* de una posición desde donde sus *bytes* no han sido previamente definidos).
* ¿Hemos encontrado un *byte* igual al actual?
* Caso Si:
  + Comparamos el siguiente *byte* desde la posición actual con el *byte* en la posición siguiente de donde encontramos un *byte* igual al primero.
  + Continuar comparando hasta que encontremos un byte que no es igual.
  + Se ha encontrado un byte que no es igual. ¿Es el número de bytes mayor que tres?
  + Caso Si:
    - Escribir el desplazamiento del PRIMER [byte](https://es.wikipedia.org/wiki/Byte) hallado y el número de bytes repetidos (tamaño).
    - Movemos el puntero a la posición con el número de bytes repetidos (porque no los hemos “salvado”) y seguimos buscando.
    - También se escribe una bandera 1.
  + Caso No:
    - Continúa la búsqueda.
* Caso No:
  + Si no se encuentra ninguna coincidencia, simplemente se escribe un *byte* sin comprimir (también se escribe un *literal* si no hay datos en la *ventana corrediza*).
  + Debe recordar poner la *bandera* a 0.

### Secuencia básica: Descompresor

* Se lee el tamaño del [archivo](https://es.wikipedia.org/wiki/Archivo_(inform%C3%A1tica)) sin [comprimir](https://es.wikipedia.org/w/index.php?title=Comprimir&action=edit&redlink=1).
* Se repite hasta que se ha [descomprimido](https://es.wikipedia.org/w/index.php?title=Descomprimido&action=edit&redlink=1) todo el [archivo](https://es.wikipedia.org/wiki/Archivo_(inform%C3%A1tica)).
* Se lee un [bit](https://es.wikipedia.org/wiki/Bit) (*la bandera*).
* Si es 0:
  + Se leen 8 *bits*, se escriben al [buffer](https://es.wikipedia.org/wiki/Buffer_de_datos) de salida (recordar que son un *byte* descomprimido) y se incrementa el [puntero](https://es.wikipedia.org/wiki/Puntero_(inform%C3%A1tica)) a la salida.
* Si es 1:
  + Se lee el desplazamiento completo (13 *bits*), luego el tamaño, copiar “tamaño” *bytes* de “desplazamiento” a la posición actual, y añadir al *puntero* a la salida “tamaño”.

**4.3 Análisis de la complejidad de los algoritmos**

Este algoritmo al implementarse de una manera básica itera por la cantidad de pixeles y luego por la cantidad de centroides, pero además esto lo debe hacer repetidas veces hasta que los centroides se mantengan constantes, este último valor es difícil de calcular, pues al ser un algoritmo sin supervisión no se puede saber bien cuando terminara, en el caso específico de fotos de 400 x 300 toma alrededor de 50 iteraciones, pero tiene una alta variación.

Tomando:

* k=cantidad de centroides
* q=cantidad de iteraciones
* M=cantidad de pixeles en x
* N=cantidad de pixeles en y

|  |  |
| --- | --- |
| **Algoritmo** | **La complejidad del tiempo** |
| Compresión | O(N\*M\*k\*q) |
| Descompresión | O(N\*M\*) |

En caso del algoritmo LZ77 la complejidad en la compresión en imágenes es O (N\*M\*k) y la descompresión es O(N\*M) donde N es la cantidad de pixels en Y, M es la cantidad de pixels en X, y K es el tamaño que tiene el buffer.

|  |  |
| --- | --- |
| **Algoritmo** | **La complejidad del tiempo** |
| Compresión | O(N\*M\*k\*) |
| Descompresión | O(N\*M\*) |

La complejidad de la compression se puede llevar hasta O(N\*M) con ayuda de otros algorimtos mas complejos, sin embargo el implementado tiene complejidad O(N\*M\*K)

# Bibliografía

[1] Castro, J. H. (2014). *Algoritmos para la Compresión de imágenes.* Bogota: Universidad Piloto de Colombia.

[2] Olive, M. S. (Septiembre de 2008). *Algoritmo de alta resolucion de imagenes sin perdidas.* Obtenido de https://tesis.ipn.mx/jspui/bitstream/123456789/5745/1/ALGORITMODECOMPR.pdf

[3] Rivera, M. E. (Diciembre de 2013). *Analisis de la compresion de imagenes utilizando clustering bajo el enfoque de colonias de hormimgas.* Obtenido de https://www.redalyc.org/pdf/816/81632390014.pdf

[4] Salcedo, L. V. (Mayo de 1994). *Compresion fractal de imagenes fijas y secuencia de imagenes utilizando algoritmos geneticos.* Obtenido de <https://repositorio.tec.mx/bitstream/handle/11285/628654/CEM337194.pdf?sequence=1>

[5] <https://programmerclick.com/article/76771868116/>

[6] <https://users.dcc.uchile.cl/~skreft/downloads/pdf/KN09.pdf>

[7] https://es.wikipedia.org/wiki/LZSS

1. [↑](#footnote-ref-2)