Universidad Autónoma de Sinaloa
Maestría en Ciencias de la Información

Propuesta y evaluación de algoritmo para la reducción en el tamaño de representación de mastografías digitales

Que como requisito parcial para obtener el grado de
Maestro en Ciencias
presenta

Vilma Sánchez López

Director de Tesis: Inés Fernando Vega López

Culiacán, Sinaloa a Febrero de 2016

Dedicatoria

Dedico esta tesis a todas las personas que me ayudaron a concretar este logro. En especial, a mis padres, Bulmaro Sánchez López y Dora Alicia López Gámez, por su apoyo incondicional, con el que siempre he contado; a mis maestros, quienes guiaron mi aprendizaje y a la Universidad Autónoma de Sinaloa, por brindarme los medios para cumplir mi meta.

Agradecimientos

Agradezco principalmente a mi asesor de tesis el Dr. Inés Fernado Vega López, por su tiempo, su dedicación y muy especialmente, por los conocimientos transmitidos. La labor de mi asesor, aunada a la de mis demás maestros de posgrado, ha hecho de mí una mejor persona: más reflexiva, y me ha brindado la posibilidad de enfrentar a los retos diarios desde una perspectiva más asertiva y optimista. Agradezco de igual manera a mis maestros y mi institución forjadora, por brindarme las herramientas necesarias para alcanzar mi tan anhelada meta.

También agradezco a mis compañeros de clase, por haber estado conmigo en momentos difíciles, por los conocimientos y retos que compartimos, por esas noches de desvelos, por tantas horas de risas y buenos momentos. Además agradezco a mi familia, por su calidez y confianza, que siempre me han brindado.

Finalmente, agradezco al Consejo Nacional de Ciencia y Tecnología por su apoyo financiero en la beca 277217. Asimismo, agradezco a la Facultad de Informática Culiacán y a la Facultad de Ciencias de la Tierra y el Espacio el apoyo brindado.

Índice general

Capítulo 1
Introducción

El cáncer de mama es la principal causa de muerte en mujeres de todo el mundo; según la Sociedad Americana Contra el Cáncer, una de cada ocho mujeres tiene riesgo de padecer cáncer de mama a lo largo de su vida (12.2 %) y una de cada 28, de morir por esta enfermedad [7]. Dicho padecimiento se origina con un tumor maligno en algunas células del seno que luego invade a células sanas [31]; la capacidad de una célula de crecer sin control e invadir otros tejidos es lo que la hace cancerosa. La mastografía digital, una imagen obtenida mediante un aparato especializado de rayos X, es el estudio preventivo más confiable para detectar el cáncer de mama. La Sociedad Americana Contra el Cáncer recomienda realizar mastografías anuales a las mujeres mayores de 40 años. Así, a medida que incrementa la cantidad de mastografías, el costo de almacenamiento y la necesidad de transmitir las imágenes de mastografía aumentan de igual manera.

Algunas instituciones médicas optan por eliminar el excedente de imágenes, otras, por comprar más espacio de almacenamiento digital. Sin embargo, existe otro problema común al que se enfrentan las instituciones médicas: en ocasiones el médico que realiza el diagnóstico no se encuentra en el mismo lugar donde se realizó el estudio, lo que genera la necesidad de trasmitir las imágenes por las redes de computadora (en otras palabras, medicina remota). El costo en tiempo de transmisión está en función del tamaño de las imágenes y del ancho de banda de la red de comunicación utilizada.

Entonces, como el tiempo de transmisión está en función del espacio de almacenamiento, existen métodos de compresión que reducen el espacio de almacenamiento y conservan la información necesaria para reconstruir la imagen en cuestión. Existen dos tipos generales de métodos de compresión de imágenes:

- Métodos de compresión con pérdida o irreversibles.
- Métodos de compresión sin pérdida o reversibles.

Los métodos de compresión sin pérdida son los más empleados en la comunidad médica y en la literatura científica son los mayormente estudiados. En comparación con los anteriores, los métodos de compresión con pérdida proporcionan un grado más alto de compresión, sin que necesariamente ello implique una pérdida de información relevante para el diagnóstico.

1.1. Definición Formal del Problema

En la revisión de la literatura científica nos percatamos de que la mayoría de los métodos de compresión se enfoca en la compresión con y sin pérdida de forma separada. Muchos de ellos se evalúan en imágenes médicas, incluidas las de mastografías; no obstante, al aplicar sólo un método de compresión a las mastografías, no se está tomando ventaja de las características particulares de las mismas. Algunas de las particularidades que tienen en común las mastografías son las siguientes: todos los estudios tienen un fondo negro y una zona de interés, la forma de la mama es similar en todas las imágenes analizadas; también, las anormalidades en los tejidos de la mama tienden a tener tonalidades más claras (en una escala de grises) que el resto de la imagen. Se requiere, pues, un método de compresión especializado en mastografías digitales que garantice la persistencia de la información relevante para el diagnóstico, que aproveche las características similares de las mastografías y que brinde una Tasa de Compresión (TC) superior a la de los métodos de compresión sin pérdida.
Con base en lo anterior, se plantea la siguiente pregunta de investigación: ¿existe una mejor forma de comprimir una mastografía digital que con los métodos de compresión optimizados para imágenes?. Nosotros partimos de la hipótesis que es posible y, para resolver esta pregunta de investigación, comenzamos por formalizar el problema de compresión de mastografías de la manera que a continuación describimos.

Sea M, una matriz de $n \times m$, la representación de una mastografía digital. Cada celda de M almacena un valor entero que se interpreta como un tono de gris en la imagen de mastografía.

Sea $f(M)$ una función que nos permite asociar un diagnóstico médico a M. Nótese que $f()$ no tiene que ser necesariamente un modelo matemático o un algoritmo, sino que la asociación puede estar dada por un humano (un experto médico). Finalmente, sea $g(M)$ una función que determina el espacio de almacenamiento digital requerido por M. Se desea encontrar una función $T()$, que transforme a M: $M' = T(M)$ de forma que $g(M') < g(M)$ y $T(M') = T(M)$.

Entonces deseamos encontrar una transformación que reduzca el espacio de almacenamiento, pero que a la vez nos permita emitir el mismo diagnóstico que podría darse a partir de una imagen con compresión reversible o con base en la propia mastografía original.

1.2. Principales Logros del Presente Trabajo

En el presente documento se plantea un método de compresión para mastografías digitales que aprovecha las características de las imágenes (nuestro objeto de estudio). A la vez, el método de compresión planteado encuentra la zona de interés, la mama, y resguarda sus valores sin ningún tipo de pérdida. El resto de la imagen es reconstruido sin tomar en cuenta su valor original; esto permite que el diagnóstico no se vea afectado tras la compresión, a la par que, en comparación con otros algoritmos similares, se reduce la cantidad de espacio requerido para almacenar las mastografías.

Como parte del presente trabajo, se realizó un estudio de los algoritmos de compresión más relevantes en la literatura científica, a fin de posteriormente poder proponer un nuevo algoritmo para la compresión de mastografías digitales, el cual integra los tipos de compresión con y sin pérdida. Nuestro algoritmo de compresión especializado en mastografías digitales explota las características particulares de éstas: tales parti-

cularidades permiten encontrar la zona de interés (la mama); el resto de la imagen es comprimido con pérdida. El algoritmo propuesto fue comparado contra otro similar y contra uno optimizado para imágenes que es bien conocido en la comunidad científica. Ahora bien, para poder medir la eficiencia de los algoritmos de compresión se requiere una medida de comparación entre dichos métodos. La mayoría de los estudios que comparan la eficiencia en materia de compresión utilizan la Tasa de Compresión (TC), es decir, la cantidad de bits necesarios para representar la imagen original entre el tamaño en bits requerido para almacenar el resultado de la compresión.
La valoración de las versiones sin pérdida y con pérdida de JPEG2000 en una base de datos pública fue parte complementaria de este estudio, necesaria para una más completa evaluación de nuestra propuesta. Como resultado de los experimentos realizados, tenemos que nuestro algoritmo logró un más alto índice de TC que el de todos los demás algoritmos de compresión de imágenes con los que fue comparado: no sólo superó al bien conocido y optimizado JPEG2000, sino que además obtuvo mejores resultados que un algoritmo híbrido similar.

1.3. Organización del Documento

El resto del documento se organiza de la siguiente manera. En el Capítulo 2 se describen los algoritmos de compresión más relevantes en la literatura científica, así como los formatos de imagen y los tipos de algoritmos de compresión aplicados en mastografías digitales. El Capítulo 3 expone la propuesta algorítmica de compresión realizada. En el Capítulo 4 se detallan los resultados obtenidos al comparar nuestra propuesta algorítmica con algoritmos de compresión con y sin pérdida. Por último, en el Capítulo 5 se presentan las conclusiones del trabajo.

Capítulo 2
Antecedentes

En este capítulo se presenta una revisión de la literatura científica especializada en temas relativos a nuestro trabajo y que fulgieron como premisa del mismo. En dicha revisión se describen los algoritmos de compresión y los formatos de imagen más utilizados, con el objetivo de mostrar un panorama amplio respecto a ambos.
Los algoritmos de compresión están divididos en dos tipos: con pérdida y sin pérdida. Se describen aquí la estructura de los algoritmos de compresión, cronología, características particulares y además se dan ejemplos del funcionamiento de cada uno de ellos. Asimismo, describimos los formatos de imagen más relevantes en la literatura científica. Lo anterior debido a que las mastografías digitales (nuestro objeto de estudio) son imágenes y pueden ser representadas con los formatos, que exponemos en estas páginas, algunos de los cuales añaden uno o más tipos de compresión. Ciertos formatos de imagen implementan algoritmos de compresión existentes y otros cuentan con un método de compresión particular diseñado especialmente para el formato de imagen del que se trate. La mayoría de los formatos de imagen soporta compresión con pérdida y sin pérdida, y algunos tienen limitación respecto a la calidad de imagen. También algunos formatos de imagen puede contener más de una imagen. Para finalizar el capítulo, describimos algunos trabajos realizados para la compresión de mastografías digitales específicamente.

2.1. Compresión

Al hablar de técnicas de compresión, en realidad nos estamos refiriendo a dos algoritmos: uno de compresión y otro de reconstrucción. Los tipos de entradas que el

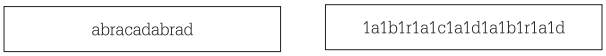
algoritmo puede recibir varían según sea el alcance del algoritmo de compresión. Sea x un archivo digital cualquiera, el algoritmo de compresión toma la entrada x y genera la salida x_c, donde x_c es un archivo digital comprimido, el cual es tomado como entrada por el algoritmo de reconstrucción para generar la reconstrucción y;
Para fines prácticos, en la literatura científica consultada y en el presente trabajo se hace referencia a los algoritmos de compresión como una técnica que incluye las dos partes antes mencionadas. Basados en el tipo de reconstrucción, los algoritmos de compresión se dividen en las siguientes dos grandes ramas [27].

- Algoritmos de compresión sin pérdida o reversibles.
- Algoritmos de compresión con pérdida o irreversibles.

La descripción de los tipos de compresión se incluye a continuación.

2.1.1. Algoritmos de Compresión sin Pérdida

La compresión sin pérdida es el tipo de compresión de datos que permite que la información original sea completamente recuperada a partir de los datos comprimidos, es decir, $x = y$. La compresión sin pérdida es generalmente empleada en aplicaciones que requieren que no haya ningún tipo de diferencia entre la información original y la reconstruida. La compresión de texto es una de las áreas en donde se utiliza la compresión sin pérdida, pues la reconstrucción exacta de cualquier texto es muy importante dado que la diferencia en una sola letra respecto al documento original puede cambiar completamente el significado de la información. La compresión de imágenes médicas utiliza, en la gran mayoría de los casos, compresión sin pérdida. Cuando hablamos de pérdida nos referimos a que x, la entrada del algoritmo compresor, no es igual a la reconstrucción, y. Las técnicas de compresión sin pérdida más relevantes en la literatura científica especializada se describen a continuación.



(a) Cadena de Entrada (b) Resultado

Figura 2.1: Ejemplo de Codificación Run-Length Encoding.

- **Codificación Run-Length**

  Esta codificación es un método simple comúnmente utilizado para la compresión de archivos, es muy eficiente cuando existen secuencias de valores repetidos. Este tipo de codificación remplaza secuencias idénticas de símbolos por el número de símbolos y el símbolo. Es de decir, los símbolos consecutivos iguales son remplazados por el símbolo y la cantidad de veces consecutivas que se repitió dicho símbolo. El método de compresión *Run-Length* es comúnmente conocido como RLE por sus siglas en inglés de *Run-Length Encoding*. No se cuenta con la fecha exacta de cuándo se comenzó a utilizar RLE, ni se sabe quién fue su creador; no obstante, existen estudios que estipulan que la compresión RLE existe desde los comienzos de la teoría de la información, en 1949 [20, 28].

  La idea detrás del método de compresión RLE es la siguiente: en caso de que el elemento d ocurra n veces consecutivas, en los datos de entrada se remplazan las n ocurrencias por el par (nd); las n ocurrencias consecutivas, de un elemento son llamadas *Run-Length* de n.

  Para ilustrar mejor el funcionamiento de RLE, veamos un ejemplo. Supongamos que se cuenta con una cadena de entrada x. Dicha cadena de entrada se puede visualizar en la Figura 2.1, en el inciso (a). Al aplicar el método de compresión obtenemos x_c, que es la versión comprimida de x.

  El resultado de la compresión lo podemos visualizar en la Figura 2.1, en el inciso (b). Para la reconstrucción sólo se tiene que realizar el proceso inverso. Para este ejemplo no se ve favorecido el método de compresión. Debido a que no hay secuencias consecutivas del mismo símbolo; al contrario aumenta la cantidad de

símbolos en lugar de disminuir.

- **Huffman**

  El método de compresión Huffman fue propuesto por David A. Huffman en 1952 [14], se trata de un método basado en frecuencias: a cada símbolo diferente en la entrada se le asigna una frecuencia que es obtenida de un preprocesamiento aplicado a la entrada x; a partir de este resultado se crea una tabla de frecuencias ordenada de mayor a menor. Posteriormente se crea un árbol binario con la tabla de frecuencias; con base en la construcción del árbol, cada símbolo tendrá un código binario Huffman único. Tal código será la representación de cada símbolo de la entrada en la salida correspondiente.

  A continuación explicamos de forma detallada el funcionamiento del algoritmo Huffman. Llamaremos t a la tabla de frecuencias. Después de tener t, se procede a la creación del árbol binario; para esto, se comienza por crear los nodos hoja del árbol binario. Los nodos hoja tienen dos elementos fundamentales: valor y frecuencia. El siguiente paso en la construcción del árbol binario es agrupar los dos nodos hoja con la menor frecuencia y agregar un nodo padre al árbol binario. Las frecuencias de estos dos nodos hoja se suman y ahora el nodo padre tendrá la suma de las dos frecuencias de los dos nodos con los menores frecuencias. La creación del árbol continúa hasta agregar el nodo raíz, el cual tendrá la suma de todas las frecuencias.

  Para entender mejor el funcionamiento del algoritmo Huffman, veamos un ejemplo. Supongamos que tenemos nuestra entrada $x = abracadabrad$, con 12 bytes de almacenamiento requeridos. Se genera una tabla de frecuencias para cada valor distinto en x. La tabla de frecuencias quedaría como se observa en el Cuadro 2.1.

  Después de aplicar el proceso antes descrito, el árbol Huffman quedaría como se observa en la Figura 2.2. Para acceder a los nodos hoja existe una

Cuadro 2.1: Tabla de Frecuencias

Valor	Frecuencia
a	5
b	2
d	2
r	2
c	1

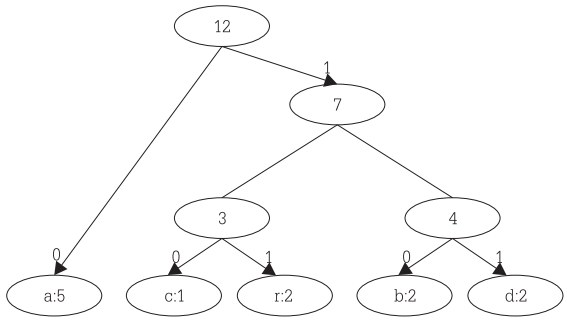


Figura 2.2: Árbol Huffman para la Cadena *abracadabrad*.

ruta de 0s y 1s, tomamos la ruta binaria de cada símbolo en la entrada y la sustituimos por cada símbolo en la entrada. Para este ejemplo la salida es $x_c = 011010101000111011010101 11$, con 4 bytes de almacenamiento requeridos.

- **LZ77**

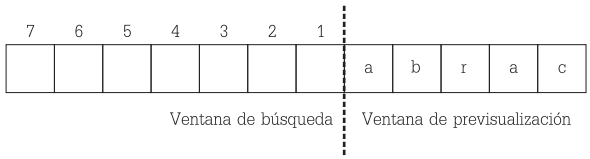
  El método de compresión LZ77, fue propuesto por Abraham Lempel y Jacob Ziv en 1977 [42], es un método de compresión basado en diccionario. Un diccionario en el ámbito de la lingüística tiene palabras ordenadas y cada una de ellas tiene una definición. Basado en esta analogía, LZ77 crea un diccionario dada una entrada, donde esta entrada puede ser cualquier archivo electrónico.

  El compresor mantiene una ventana desplazante que es considerada como el área de trabajo. Dicha ventana está dividida en dos secciones, la parte de búsqueda y la parte de previsualización. La sección de búsqueda contiene valores del diccionario, los más recientemente codificados. Por otra parte, la sección de

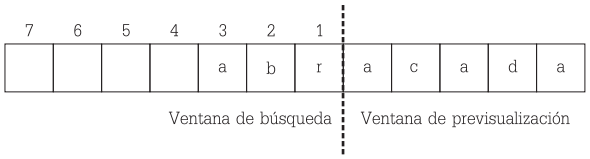
previsualización mantiene una parte de la cadena de entrada. El tamaño de la ventana de búsqueda y el de la ventana de previsualización son valores configurables. Cuando el compresor encuentra coincidencias de la ventana de búsqueda con la ventana de previsualización, las guarda en el diccionario.

LZ77 busca coincidencias del primer elemento de la ventana de previsualización con la ventana de búsqueda y si encuentra coincidencias verifica qué tan larga es la coincidencia; para lograrlo LZ77 mantiene un apuntador, p, que almacena la dirección de la coincidencia más larga y continúa revisando en toda la ventana de búsqueda: si LZ77 encuentra coincidencias, guarda en el diccionario la más larga. Para guardar un nuevo elemento en el diccionario, el algoritmo utiliza la tripleta $< o, l, c >$, donde o es la distancia desde p hasta el inicio de la ventana de previsualización, l es la longitud de la coincidencia y c es el valor que representa el símbolo inmediato posterior a la coincidencia en la ventana de previsualización. En caso de no encontrarse coincidencia, c toma el valor del primer elemento de la ventana de previsualización mientras que o y l toman el valor de 0. El inicio de la ventana de previsualización se desplaza $l+1$ símbolos.

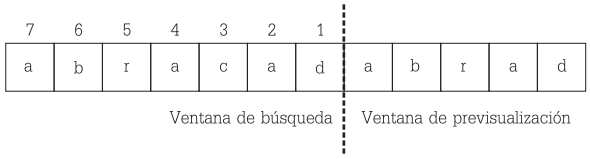
Para entender mejor este proceso, veamos un ejemplo. Supongamos que tenemos la cadena de entrada $x = abracadabrad$. La longitud de la ventana de búsqueda es de 7 y la ventana de previsualización es de 5. En este punto, nuestra ventana de búsqueda se encuentra vacía y la ventana de previsualización tiene los primeros 5 elementos de la cadena de entrada. Esta configuración inicial se puede observar en la Figura 2.3, en el inciso (a). Después la búsqueda de subcadenas de la ventana de previsualización en la ventana de búsqueda comienza. Debido a que el diccionario está vacío, los tres primeros valores (a,b,r) no se encuentran en él y son directamente agregados al diccionario. El siguiente elemento en la ventana de previsualización es a y se encuentra en la ventana de búsqueda. Como sólo hay una coincidencia y es de longitud uno, se agrega al diccionario el



(a) Estado Inicial



(b) Después de 3 Iteraciones



(c) Salida Final

Figura 2.3: Diferentes Estados de la Ventana Desplazante Creada por el Algoritmo LZ77 con la Cadena de Entrada *abracadabrad*.

cuarto elemento, que es la tripleta (3,1,c), donde el primer valor es el apuntador a la cadena más larga encontrada, el segundo valor es la longitud de la coincidencia (en este caso la longitud es uno) y el último valor del elemento agregado al diccionario es el siguiente símbolo de la ventana de previsualización, que es c. Se desplaza la ventana de previsualización $l+1$ elementos; el diccionario continúa creándose bajo este mismo proceso. La salida (que es el diccionario) del algoritmo LZ77 para este ejemplo se puede visualizar en la Figura 2.4.

Para efectuar la reconstrucción de la cadena de entrada mediante la salida (diccionario), el algoritmo reconstructor comienza tomando el primer elemento del diccionario, la tripleta (0,0,a). Los dos primeros elementos de la tripleta tienen valor de 0, entonces el tercer valor de la primera tripleta del diccionario es agregado a la cadena de reconstrucción: $y = a$. Lo mismo sucede para las dos siguientes tripletas del diccionario; hasta este punto, nuestra cadena de reconstrucción es $y = abr$. En la cuarta tripleta del diccionario, sus valores o y

Salida LZ77
(0,0,a)
(0,0,b)
(0,0,r)
(3,1,c)
(2,1,d)
(7,4,d)

Figura 2.4: Diccionario Creado por el Algoritmo LZ77.

l son diferentes de 0, en este caso, el algoritmo busca dentro de y la subcadena de coincidencia desde o hasta l valores, a partir del último valor agregado a y y agrega la subcadena de coincidencia seguida del valor de c en la tripleta en cuestión: ahora, $y = abrac$. Este proceso continúa hasta recorrer todas las tripletas del diccionario y, como resultado, tenemos que y es una copia idéntica de x.

- **LZSS**

  LZSS es un método de compresión basado en LZ77, fue propuesto por James A. Storer y Thomas G. Szymanski en 1982 [33]. Las mejoras de este método respecto a LZ77 consisten en modificar la ventana de previsualización en una cola circular, mantener el diccionario (la ventana de búsqueda) en una estructura de árbol binario; además como parte de las mejoras en cuanto a rendimiento, LZSS guarda las coincidencias del diccionario en elementos con solo dos valores. La búsqueda de coincidencias es igual que la de LZ77, salvo por la forma de almacenamiento: en caso de no encontrar coincidencias mayores a longitud 1, el algoritmo almacena los datos sin comprimir; de encontrar coincidencias, se almacena $< o, l >$, donde o es la distancia desde el apuntador hacia la ventana de previsualización y l es la longitud de la coincidencia (al igual que LZ77). Para saber cuándo un elemento pertenece a un código $< o, l >$ o a un elemento

sin comprimir, cada elemento del diccionario es antecedido por un bit 0 ó 1, 0 indica que se trata de un elemento sin comprimir y 1 indica que el elemento es de tipo código $< o, l >$.

Para ilustrar mejor la forma de codificación de LZSS, veamos cómo quedaría la cadena de entrada $x = abracadabrad$. Primero aplicamos el mismo método de búsqueda de subcadenas repetidas utilizado por LZ77, obtenemos como resultado $x_c = [0a][0b][0r][0a][0c][0a][0d][1(7,4)][0d]$. Para reconstruir x_c, los elementos que comienzan con el primer bit con valor 0 son reconstruidos tal cual y los elementos que son antecedidos por bit 1 son remplazados por su cadena correspondiente. Cabe mencionar que los corchetes y los paréntesis incluidos en x_c son meramente ilustrativos. Cada uno de los elementos del diccionario es codificado en una estructura de árbol binario.

- **LZ78**

  LZ78 es un algoritmo de compresión basado en diccionario, fue propuesto por Abraham Lempel y Jacob Ziv [43] un año después de que publicaran LZ77. LZ77 y LZ78 comparten características similares, la diferencia radica en la construcción del diccionario y la forma de referirse a los elementos previamente codificados. Para empezar LZ78 no tiene limitantes en cuanto a espacio de búsqueda o de coincidencias. LZ78 a diferencia de LZ77, no envía el diccionario creado al descompresor, lo que envía al descompresor es una codificación que sirve para que el descompresor reconstruya el diccionario y así generar la reconstrucción de la entrada. LZ78 crea un diccionario y una salida. Cada elemento de la salida está compuesto por dupla $< p, s >$. Donde p es el apuntador al diccionario de la coincidencia y s es el carácter inmediato siguiente no encontrado en el diccionario. Los elementos del diccionario están compuestos a su vez por dos elementos: una cadena y un apuntador a esa cadena. Las cadenas son patrones creados a partir de la entrada. La construcción del diccionario y la salida es simultanea.

Apuntador	Valor
1	a
2	b
3	r
4	ac
5	ad
6	ab
7	ra
8	d

Figura 2.5: Diccionario Creado por LZ78 para la Entrada *abracadabrad*.

LZ78 recorre la cadena de entrada en busca de coincidencias en el diccionario. Por cada elemento de la entrada LZ78 busca a partir del elemento en cuestión, la subcadena más larga de x encontrada en el diccionario. Posteriormente guarda en la salida el índice de la coincidencia y el elemento inmediato siguiente de la subcadena en la entrada. En caso de no encontrar coincidencias LZ78 guarda en la dupla de salida un 0 y el elemento no encontrado en el diccionario. Además agrega el elemento no encontrado como un nuevo elemento del diccionario.

Para ilustrar mejor el funcionamiento de LZ78 veamos un ejemplo. Supongamos que tenemos la cadena de entrada $x = abracadabrad$, la variable de salida es x_c. El algoritmo revisa si el elemento a se encuentra en el diccionario. Como a es el primer elemento, no se encuentra en el diccionario; entonces, LZ78 lo agrega al diccionario y también a la salida. Lo mismo sucede con los siguientes tres elementos, son agregados al diccionario y a la salida. El siguiente elemento a se encuentra en el diccionario pero ac no, c es el siguiente elemento en la entrada, entonces se agrega al diccionario ac y a la salida se agrega la dupla $< l, c >$. Este proceso continúa hasta terminar de agregar todos los elementos al diccionario. Al final el diccionario para la entrada x quedaría como se observa en la Figura 2.5 y la salida como se observa en la Figura 2.6.

Salida LZ78
(0,a)
(0,b)
(0,r)
(1,c)
(1,d)
(1,b)
(3,a)
(0,d)

Figura 2.6: Salida del Algoritmo LZ78 para la Entrada *abracadabrad*.

La reconstrucción de y mediante la entrada x_c es similar a la forma en que LZ77 lo hace. LZ78 toma cada elemento de la entrada y si el elemento l de la dupla en cuestión es 0 entonces agrega a y el valor de c. De lo contrario si el el elemento l de la dupla en cuestión es diferente de 0, LZ78 agrega a la salida el valor apuntado de l en el diccionario, más el valor de c de la dupla en cuestión; a la par LZ78 crea el diccionario; donde cada dupla en x_c es agregada al diccionario como un nuevo elemento. De esta forma los elementos que se agreguen al diccionario serán posteriormente referenciados en las duplas de x_c. Finalmente nos quedamos con y que es una copia exacta de x y el diccionario es descartado.

- **Lempel-Ziv-Welch**

  LZW (Lempel-Ziv-Welch) es también un algoritmo de compresión basado en diccionario y fue desarrollado por Terry Welch en 1984 [38] como una mejora de la propuesta realizada por Abraham Lempel and Jacob Ziv en 1978 [43]. A partir de los datos de entrada x, LZW crea un diccionario que tiene dos elementos fundamentales: los valores y los apuntadores a esos valores; éstos son agregados al diccionario de forma ordenada y su posición en el diccionario es la clave para una correcta reconstrucción. A cada par (apuntador, valor) lo lla-

maremos elemento del diccionario. A la vez que crea el diccionario, el algoritmo LZW genera una salida x_c, que será la entrada del algoritmo reconstructor.

Lo primero que hace LZW es inicializar el diccionario con todos los valores individuales posibles de x. En el caso común de símbolos de 8 bits, las primeras 256 entidades del diccionario son ocupadas antes que cualquier otro valor. Después de inicializar el diccionario, el algoritmo comienza la búsqueda de valores. El primer valor de la entrada x siempre será encontrado en el diccionario, puesto que fueron precargados todos los valores posible de longitud uno.

LZW prosigue el recorrido de x en busca de la subcadena de x más larga no encontrada en el diccionario. Llamaremos e al valor en cuestión de la cadena de entrada. El algoritmo crea una variable temporal, t, que mantiene una subcadena de x no encontrada en el diccionario. El algoritmo valida que $t + e$ se encuentren en el diccionario, si esta condición se cumple, entonces, t obtiene el valor de $t + e$. La búsqueda continúa hasta no encontrar $t + e$ en el diccionario; una vez que sucede, se escribe en x_c el apuntador al valor t en el diccionario.

En seguida, se agrega un nuevo elemento al diccionario con el nuevo valor $t + e$ y t toma el valor de e; este proceso se repite hasta terminar de recorrer x. La cantidad de iteraciones que el algoritmo LZW va a realizar es la longitud de los valores de entrada más uno.

A continuación ejemplificamos el funcionamiento del algoritmo LZW. Supongamos que tenemos la cadena de entrada $x = abracadabrad$. Para este caso, los valores iniciales por agregar al diccionario son a, b, c, d y r ya que constituyen el alfabeto de la cadena. El diccionario, hasta este punto, cuenta con los cinco primeros elementos mostrados en la Figura 2.7, t tendría el valor de cadena vacía y $e = a$. La búsqueda comienza con $t + e$. Es decir, se verifica si $t + e$, se encuentra en el diccionario. Como todos los valores posibles de longitud uno fueron precargados al diccionario, $t + e$ que es igual a a, se encuentra en el dic-

Apuntador	Valor
1	a
2	b
3	c
4	d
5	r
6	ab
7	br
8	ra
9	ac
10	ca
11	ad
12	da
13	abr
14	rad

Figura 2.7: Diccionario Creado por el Algoritmo LZW con una Cadena de Entrada *abracadabrad*.

cionario en la posición 1. Debido a que $t+e$ se encontró en el diccionario, ahora nuestra variable t tiene el valor de e, es decir, $t = a$ y $e = b$. LZW prosigue la búsqueda de $t + e$ en el diccionario, ab no se encuentra en el diccionario; entonces, se agrega a la salida el apuntador que representa el valor de entrada a en el diccionario. En el ejemplo, este apuntador es 1 y se agrega la subcadena ab como otro elemento al diccionario; t adquiere el valor de e, es decir, $t = b$ y e adquiere el siguiente valor en la cadena de entrada, que es r.

Ahora en el diccionario se tienen los 6 primeros elementos de la Figura 2.7. De esta forma se recorre cada uno de los valores de x hasta encontrar el final de la cadena de entrada. Por último, cuando el algoritmo encuentra que la cadena de entrada terminó, agrega al archivo de salida x_c el apuntador del último elemento de la cadena, en este caso 4, que es el apuntador de d, el último valor en la cadena para nuestro ejemplo. El diccionario creado por LZW se puede observar en la

Figura 2.7 y la salida para este ejemplo es 1251314684.

El algoritmo descompresor toma como entrada x_c y reconstruye el diccionario. Para ilustrar el mecanismo de funcionamiento del descompresor procederemos a descomprimir x_c. Se sabe que cada elemento de x_c es un índice al diccionario. Los elementos en cuestión de x_c siempre serán encontrados en el diccionario y a partir de ellos se crearán los siguientes elementos del diccionario. Se tienen la variable e, que es el elemento en cuestión en x_c; ev, que almacena el valor correspondiente de e en el diccionario; $ev1$, que es el primer valor de ev; la variable y, que representa la salida; y usaremos t como variable temporal.

El algoritmo comienza por precargar los símbolos distintos en el alfabeto, al igual que lo hace el algoritmo compresor. Hasta este punto, tenemos los cinco primeros elementos del diccionario. La configuración de variables es la siguiente: $e = 1$, $ev = a$, $ev1 = a$, $y = a$ y t se encuentra vacía. Posteriormente se verifica si $t + ev1$ se encuentra en el diccionario, de ser así, se prosigue con el análisis del siguiente elemento de x_c y $t = ev$; de lo contrario; de no encontrarse $t + ev1$, este es agregado al diccionario como un nuevo elemento y $t = ev$.

En la segunda iteración la configuración de variables es la siguiente: $e = 2$, $ev = b$, $ev1 = b$, $t = a$ y $y = ab$; en este caso, $t + ev1 = ab$ no se encuentra en el diccionario por lo tanto es agregado al diccionario como otro elemento. Este proceso continua por cada elemento en x_c hasta terminar de analizar todos los elementos de x_c.

- **Codificación RAR**

  RAR, por sus siglas en inglés de *Roshal ARchive*, es un algoritmo de compresión basado en diccionarios, fue creado por Eugene Roshal en 1993 [11]. RAR está basado en los algoritmos de compresión sin pérdida LZSS y LZ77; al igual que estos, RAR utiliza una ventana deslizante cuyo tamaño puede variar entre 64KB y 4MB. Como mencionamos en la descripción de los algoritmos LZSS y

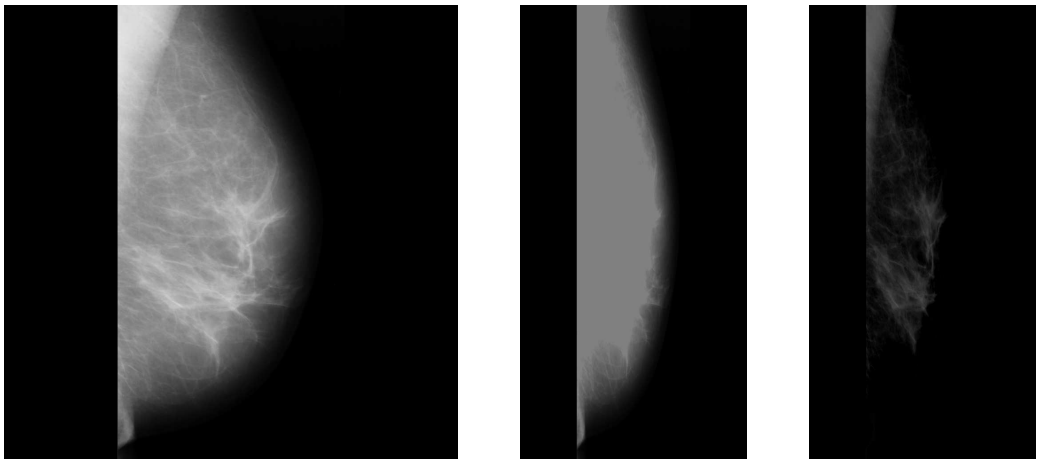
LZ77, estos buscan subcadenas de la ventana de previsualización en la ventana de búsqueda; RAR limita la longitud mínima de coincidencia de las subcadenas a dos unidades; también existen códigos especiales que son usados para mejorar la compresión de posiciones repetidas. Valores, desplazamientos y longitudes de coincidencia se comprimen con un algoritmo de compresión sin pérdida Huffman. Además RAR integra completamente la compresión de varios archivos a la vez. RAR es un algoritmo propietario, la información que plasmamos aquí fue la obtenida de las fuentes públicas proporcionadas por el desarrollador, por dicha razón se vio limitada la descripción.

2.1.2. Algoritmos de Compresión con Pérdida

Este tipo de algoritmos de compresión proporciona grados más altos de compresión que los algoritmos de compresión sin pérdida. La compresión con pérdida es ampliamente utilizada cuando la reconstrucción de la información sólo necesita ser razonablemente similar a la original [30]. Las principales técnicas utilizadas por este tipo de compresión se describen a continuación.

- **Transformada Wavelet**

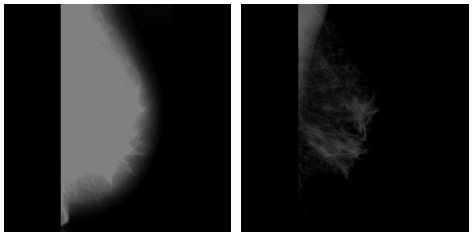
  Comúnmente la Transformada Wavelet se utiliza como procesamiento previo para comprimir señales digitales, entre las cuales se encuentran las imágenes, que son señales de dos dimensiones. Supongamos que se tiene una imagen de tamaño $n \times m$, donde n es el número de renglones y m es el número de columnas de la imagen. Aplicamos una función Wavelet a los renglones y obtendremos dos imágenes de tamaño $n \times \frac{m}{2}$ cada una. La primera imagen es el resultado de la transformación y la segunda consiste en los errores correspondientes respecto al valor en la imagen original y el resultado de la transformación. Posteriormente, se aplica de nuevo la función Wavelet a las columnas y obtendremos



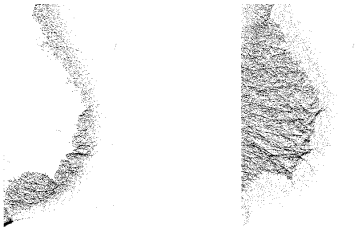
(a) Imagen Original (b) L (c) H

Figura 2.8: Aplicación del Primer Filtro Wavelet a los Renglones de una Mastografía con una Función Haar.

cuatro imágenes de $\frac{n}{2} \times \frac{m}{2}$ cada una. Cada vez que se aplica la función Wavelet se obtienen dos imágenes: una imagen con los coeficientes obtenidos de la transformación y otra con el error de los valores obtenidos respecto a los valores originales. Los segundos coeficientes (el error) aportan menor información y son considerados como los coeficientes de refinamiento. Hasta este punto, tenemos los coeficientes de las cuatro imágenes resultado; juntas forman el primer nivel de descomposición de la Transformada Wavelet. En la Figura 2.8 podemos observar el resultado de aplicar la Transformada Wavelet Haar a los renglones de una imagen de mastografía digital. Los coeficientes básicos (L) de la imagen de mastografía utilizada como ejemplo se pueden observar en la Figura 2.8 (b) y los coeficientes de refinamiento (H), en la Figura 2.8 (c); en la imagen utilizamos L (coeficientes básicos) para los coeficientes que son el resultado de aplicar la Transformada Wavelet y H para los coeficientes de refinamiento. Utilizamos H y L para referirnos a los coeficientes con el fin de mantener la nomenclatura comúnmente empleada en la literatura especializada proveniente del inglés *Low and High Frequency Coefficients* [5].



(a) LL (b) HL



(c) LH (d) HH

Figura 2.9: Transformación de una Imagen de Mastografía en los Diferentes Niveles de Descomposición.

Si la función Wavelet se aplica primero a los renglones y luego a las columnas, nos quedaremos con una matriz de coeficientes. Los coeficientes que aportan más información son los de la esquina superior izquierda (LL), que representan un cuarto del tamaño de la imagen original.

En la Figura 2.9 mostramos las imágenes resultantes del primer nivel de descomposición de la Transformada Wavelet. En las Figuras 2.9 (c) y 2.9 (d) los valores de los píxeles negros fueron invertidos por blancos; esto, con el fin de poder visualizar mejor los píxeles con valores cercanos a 0 (negro) y evitar que se pierdan en el fondo negro.

El proceso de descomposición anteriormente descrito se aplica recursivamente a los coeficientes básicos (LL) hasta que LL sea de tamaño 1×1. A cada iteración del proceso de descomposición se le conoce como nivel de descomposición n. En esta sección acabamos de describir el primer nivel de descomposición de la

Transformada Wavelet.

La Transformada Wavelet en sí no efectúa compresión, la compresión se obtiene mediante la eliminación de coeficientes. La razón por la que acabamos de describir la Transformada Wavelet es porque los siguientes algoritmos la utilizan para realizar compresión. Cada uno de los métodos de compresión que utilizan la Transformada Wavelet, maneja y descarta los coeficientes de diferentes formas. Hasta aquí, es importante tener claros los términos de coeficientes y niveles de descomposición que utilizaremos en las siguientes secciones.

- **JPEG**

  Joint Photographic Experts Group (JPEG) en la actualidad es el estándar de compresión de imágenes más utilizado, fue creado en 1992 por el comité que lleva el mismo nombre [14]. En caso de que la imagen por comprimir sea a color, JPEG crea una imagen de 24 bits de profundidad, 8 bits para cada canal (rojo, verde y azul); para las imágenes a escala de grises, crea una imagen de 8 bits de profundidad. Dicho estándar de compresión utiliza la Trasformada Discreta de Coseno (TDC). JPEG permite compresión con y sin pérdida, aunque la versión sin pérdida no es muy utilizada, de hecho, la mayoría de los implementadores no la incluyen en su desarrollo. Una de las principales características de JPEG es su flexibilidad a la hora de ajustar la tasa de compresión: se puede ajustar la pérdida al nivel deseado, considerando que a una mayor pérdida corresponde una menor calidad en la reconstrucción.

  JPEG es un método de compresión muy amplio que se puede revisar en las especificaciones del formato [14]; pero, por cuestiones de espacio y enfoque, en este documento sólo revisaremos el tipo de compresión con pérdida basado en la TDC sobre imágenes a escala de grises. Esto, debido a que nuestro tema de estudio son las imágenes de mastografía y su representación se encuentra, precisamente, en escala de grises.

El primer paso del algoritmo consiste en organizar los píxeles de la imagen en grupos de 8×8. Cada grupo de datos es llamado *unidad de datos*, y cada unidad de datos es comprimida por separado. Si el número de filas de la imagen o de columnas no es múltiplo de 8, la fila de abajo y la columna de la derecha son replicadas las veces que sea necesario para obtener un total divisible entre 8.

La TDC es aplicada posteriormente a cada unidad de datos para crear un mapa de frecuencias de 8×8. La naturaleza de la función de coseno utilizada por la TDC implica pérdida debido a la limitada precisión de la aritmética computacional. Esto significa que aun sin el siguiente paso, que es donde se da la mayor pérdida, habrá algún tipo de pérdida en calidad de imagen, aunque es normalmente pequeña y no significativa.

Las unidades de datos transformadas constan de 64 coeficientes, resultado de la transformación TDC. Cada uno de los 64 coeficientes es dividido entre un número entero. Dicho número entero puede ser generado al azar, o mediante una función. Sin embargo, el estándar de compresión JPEG propone tablas de cuantificación de 8×8 para obtener los dividendos, donde cada coeficiente será dividido por su correspondiente dividendo en la tabla de cuantificación.

Después de la división, el resultado de dividir los coeficientes por los valores correspondientes de la tabla de cuantificación, éstos son redondeados a entero, es aquí donde la información es perdida irreversiblemente. Los coeficientes de cuantificación de gran magnitud causan mayor pérdida. Cada uno de los 64 valores de cuantificación son parámetros de JPEG y pueden, en principio, ser especificados por el usuario. El resultado es codificado por una combinación del método de compresión Huffman y *Run-Length Encoding*. El acomodo de los bloques de datos comprimidos dependerá del modo de compresión. Las tablas de cuantificación pueden ser o no agregadas a los datos comprimidos. En la Figura 2.10 se puede visualizar la estructura general del algoritmo de compresión JPEG.

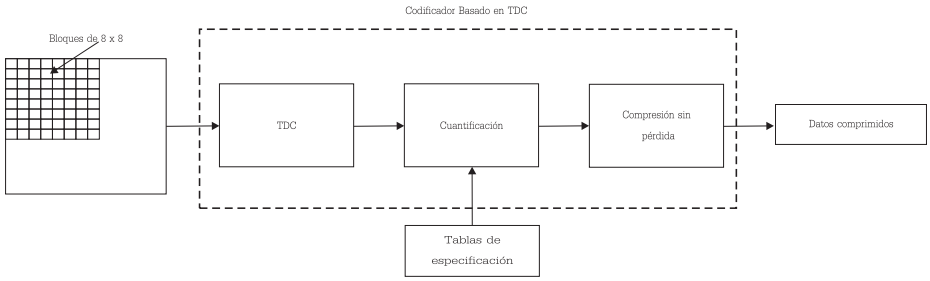


Figura 2.10: Esquema General del Algoritmo Compresor JPEG.

Cuadro 2.2: Bloque de Entrada de 8×8 Perteneciente a una Imagen en Escala de Grises.

247	243	231	215	203	181	153	129
250	248	243	231	215	203	177	157
250	248	243	231	215	203	177	156
249	248	243	231	215	202	176	156
249	247	242	230	214	202	176	156
249	247	242	230	214	201	175	155
248	246	241	229	213	201	175	155
248	246	241	229	213	201	175	154

Para dejar más claro el funcionamiento de JPEG, veamos un ejemplo: supongamos que estamos procesando una imagen en escala de grises y tenemos un bloque de 8×8 con los valores de los píxeles que se muestran en el Cuadro 2.2.

El primer paso consiste en aplicar la TDC a los 64 valores del Cuadro 2.2, los resultados de la transformación los podemos observar en el Cuadro 2.3. Siguiendo los pasos del algoritmo, JPEG aplica la cuantificación: los resultados se pueden observar en el Cuadro 2.4 y el redondeo de los resultados lo podemos visualizar en el Cuadro 2.5. La tabla de cuantificación utilizada para este ejemplo se muestra en el Cuadro 2.6. Finalmente, aplicamos un método de compresión con pérdida como el estándar lo especifica, para este caso, empleamos *Run-Length Encoding*. El resultado comprimido para nuestro ejemplo es el siguiente:

Cuadro 2.3: Resultados de Aplicar la Transformada Discreta de Coseno (TDC).

703.5712	697.5608	680.9438	645.5885	601.7479	563.5641	489.3179	430.628
0.6189	-0.2644	-3.6971	-5.6587	-3.6971	-8.4208	-9.4016	-11.1506
-1.5772	-2.963	-6.1966	-8.0443	-6.1966	-10.1627	-11.0866	-13.2049
-1.1012	-2.2105	-5.1206	-6.7836	-5.1206	-9.4906	-10.3220	-11.0789
-1.4142	-1.7678	-4.2426	-5.6569	-4.2426	-7.7782	-8.4853	-9.8995
-1.161	-1.3009	-3.2454	-4.3565	-3.2454	-6.3414	-6.897	-7.8279
-0.1121	-0.6861	-2.0255	-2.7909	-2.0255	-4.2095	-4.5922	-6.0108
-0.2374	-0.9229	-1.6057	-1.9959	-1.6057	-1.675	-1.8701	-2.5785

1[44]1[63]1[68]1[40]1[5]1[4]2[3]56[0], donde cada valor de la tabla (los valores encerrados entre corchetes) es antecedido por su frecuencia. Los corchetes no forman parte de la compresión, los hemos utilizado en este ejemplo sólo para poder distinguir el valor y su frecuencia. Las implementaciones de *Run-Length Encoding* manejan de diferente forma las frecuencias y los valores; sin embargo, las implementaciones de los métodos de compresión no son temas que abordemos en en este trabajo.

Cuadro 2.4: Resultados de la Cuantificación con una Tabla de Cuantificación Propuesta por el Estándar JPEG.

43.9732	63.4146	68.0944	40.3493	4.8528	4.0255	3.2405	2.6747
0.0516	-0.022	-0.2641	-0.2978	-0.0293	-0.0533	-0.0588	-0.0719
-0.1127	-0.2279	-0.3873	-0.3352	-0.0443	-0.0647	-0.0656	-0.0846
-0.0787	-0.13	-0.2328	-0.2339	-0.0339	-0.0508	-0.0573	-0.0684
-0.0786	-0.0804	-0.1147	-0.101	-0.0253	-0.0714	-0.0824	-0.0559
-0.0484	-0.0372	-0.059	-0.0681	-0.0179	-0.061	-0.061	-0.0408
-0.0023	-0.0107	-0.026	-0.0321	-0.0197	-0.0348	-0.0383	-0.0595
-0.0033	-0.01	-0.0169	-0.0204	-0.0143	-0.0168	-0.0182	-0.013

44	63	68	40	5	4	3	3
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Cuadro 2.5: Coeficientes Redondeados Después de Haber Realizado la División con los Valores de la Tabla de Cuantificación.

16	11	10	16	124	140	151	161
12	12	14	19	126	158	160	155
14	13	16	24	140	157	169	156
14	17	22	29	151	187	180	162
18	22	37	56	168	109	103	177
24	35	55	64	181	104	113	192
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	199

Cuadro 2.6: Tabla de Cuantificación del Estándar JPEG para Imágenes a Escala de Grises.

Por otro lado, el proceso de decodificación de JPEG es la inversa del codificador. Primeramente se descomprime el resultado. Posteriormente los resultados se multiplican por los valores correspondientes en la tabla de cuantificación. Después a estos valores se les aplica la TDC a la inversa (el resultado de la inversa de la TDC lo podemos observar en el Cuadro 2.7 y, en nuestro caso, para el ejemplo utilizado, el resultado de aplicar el decodificador lo podemos observar en el Cuadro 2.8). La decodificación presenta pérdida en la reconstrucción, no obstante, es similar a los datos originales·

Cuadro 2.7: Resultado de Aplicar la Inversa de la TDC a los Valores del Cuadro 2.5.

704	693	680	640	620	560	453	483
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Cuadro 2.8: Resultado de la Descompresión de JPEG.

248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77
248.90	245.01	240.42	226.27	219.20	197.99	160.16	170.77

- **Compresión por EZW**

  EZW, por sus siglas en inglés de *Embedded Zerotree Wavelet*, fue creado por Jerome M. Shapiro en 1993 [29]. EZW es un método de compresión adaptado para imágenes y está basado en Wavelet. Como mencionamos anteriormente, al utilizar la Transformada Wavelet en señales bidimensionales obtenemos una matriz de coeficientes ordenados, de tal forma que el coeficiente que aporta mayor información comúnmente se encuentra en la esquina superior izquierda.

  Los coeficientes se dividen en niveles de transformación, como lo mencionamos anteriormente. EZW acomoda los coeficientes en una estructura tipo árbol, donde cada nivel de nodos del árbol corresponde a un nivel de transformación. La raíz del árbol siempre es la esquina superior izquierda de la matriz de coeficientes (el último nivel de transformación). Los hijos de este nodo serán los elementos del siguiente nivel de transformación y así sucesivamente. La última capa de nodos estará compuesta por los coeficientes que conforman el primer nivel de transformación. El orden de acomodo de cada uno de los coeficientes de cada capa es: primero la esquina superior derecha de la capa de nodos en cuestión, esquina inferior izquierda y finalmente los coeficientes de la esquina inferior derecha.

  En muchos casos los coeficientes cercanos a la raíz del árbol tienen magnitudes mayores que los más alejados de la raíz. Entre más alto es el valor del coeficien-

te (sin importar el signo), mayor es la cantidad de información que aporta a la reconstrucción de la imagen. EZW permite establecer un umbral. Dicho umbral nos sirve para clasificar los coeficientes significantes y los insignificantes. Dado un umbral T, si un coeficiente tiene una magnitud mayor a T, es considerado un coeficiente significante al nivel de T. Si la magnitud del coeficiente es menor que T, es considerado insignificante y, si además todos sus descendientes también tienen magnitudes menores a T, es considerado como *zerotree root*. Ahora bien, puede suceder que el coeficiente sea menor que T, pero algunos de sus descendientes tengan un valor mayor que T. Este tipo de coeficiente es llamado *isolated zero*. Los coeficientes significantes son los que se conservan, los demás son descartados. Los nodos del árbol pueden tener cuatro categorías. A cada nodo se le asigna una etiqueta, por así decirlo. Las categorías pueden ser: coeficiente positivo, coeficiente negativo, *zerotree root*, *isolated zero* y *isolated zero*. Los descendientes de los nodos que son etiquetados como *zerotree root* son eliminados de la estructura de árbol.

Cuadro 2.9: Coeficientes Wavelet para Representar Ejemplo de EZW.

26	6	17	10
-7	7	6	4
4	-4	4	-3
2	-2	-2	0

Para tener más claro el funcionamiento del algoritmo, veamos un ejemplo: supongamos que tenemos una matriz de coeficientes Wavelet de 4×4 (ver Cuadro 2.9) y un umbral de $T = 16$. El primer paso por realizar es identificar los coeficientes significantes al nivel de T y convertir en 0 los que no lo son. Posteriormente, convertimos la matriz de coeficientes resultante del paso anterior en un árbol EZW. La creación del árbol EZW a partir de la matriz de coeficientes es muy sencilla. De ante mano sabemos que la Transformada Wavelet tiene niveles de transformación. Llamaremos ln al último nivel de transformación y

$l1$ al primer nivel de trasformación. La creación del árbol comienza desde la raíz. Este valor siempre será la esquina superior izquierda de la matriz de coeficientes del nivel de transformación ln. Para nuestro ejemplo, la raíz del árbol es 26. Los descendientes del nodo raíz son los demás coeficientes del nivel de trasformación ln. Estos coeficientes, para nuestro ejemplo particular, son 6,-7 y 7. Nuestro ejemplo sólo tiene dos niveles de trasformación, entonces, nuestro árbol sólo tiene tres capas de nodos. El árbol completo lo podemos observar en la Figura 2.11.

Los coeficientes 7 y -7 son etiquetados como *zerotree root* debido a que su valor es inferior a T y todos sus descendientes lo son de igual manera. En el caso del nodo con valor 6 de la misma capa, su valor es menor que T, pero tiene un descendente que es mayor que 16. Este tipo de coeficientes es etiquetado como un coeficiente del tipo *isolated zero*. Por otro lado, los descendientes de los nodos considerados *zerotree root* son eliminados del árbol. Finalmente, las etiquetas y el árbol modificado son comprimidos con un método de compresión basado en entropía.

El descompresor EZW toma como entrada un archivo comprimido (el resultado del algoritmo compresor EZW) y lo descomprime con el algoritmo de compresión previamente utilizado por el algoritmo de compresión EZW. Enseguida, el descompresor EZW reconstruye la matriz de coeficientes a partir de la estructura de árbol. Naturalmente que los descendientes de los nodos etiquetados como *zerotree root* serán reconstruidos como coeficientes con valor de 0. El algoritmo descompresor reubica cada uno de los coeficientes en la matriz de coeficientes a partir del árbol, siguiendo la misma lógica de creación de éste. Así, tenemos ya los coeficientes listos para aplicar la inversa de la Transformada Wavelet y poder reconstruir nuestra imagen.

- **Compresión por SPIHT**

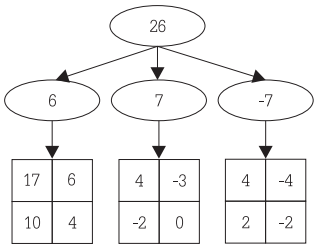


Figura 2.11: Árbol Generado por EZW a partir de una Matriz de Coeficientes Wavelets.

SPIHT, por sus siglas en inglés de *Set Partitioning in Hierarchical Trees*, es una generalización de EZW. SPIHT fue propuesto por Said y Pearlman en 1996 [26]. Al igual que EZW, SPIHT utiliza la matriz de coeficientes generada por la Transformada Wavelet. Sin embargo, SPIHT divide los coeficientes en tres categorías: coeficientes significantes, coeficientes insignificantes y conjuntos de coeficientes insignificantes. SPIHT ordena los coeficientes de mayor a menor y crea una estructura de árbol. Los coeficientes significantes son los que tienen una magnitud mayor a un umbral T, donde T es un umbral establecido por el usuario o por el mismo algoritmo. El árbol creado por SPIHT guarda en una partición las coordenadas de los coeficientes significantes, en otra partición guarda los coeficientes insignificantes y en otra partición los conjuntos de coeficientes insignificantes.

El descompresor de SPIHT hace uso de la organización jerárquica de los coeficientes para descomprimir y mostrar primero el resultado de los coeficientes más significantes hasta los menos significantes.

- **JPEG2000**

  JPEG2000 es un estándar de compresión con y sin pérdida de datos [1]; surgió como una propuesta de mejora para el algoritmo JPEG en el 2000. JPEG2000 emplea la Trasformada Wavelet, mientras que JPEG emplea TDC. JPEG2000 en un estándar de compresión de tipo progresivo en cuanto a la

tasa de compresión con pérdida. Al igual que JPEG, la calidad en la reconstrucción de la imagen es ajustable. JPEG2000 además permite compresión por zonas de interés. JPEG2000 tiene una mayor complejidad computacional, pero proporciona mayor calidad en la compresión, comparado con JPEG.

Los pasos fundamentales del algoritmo son los siguientes: JPEG200 comienza dividiendo la imagen en rectángulos no traslapados. Este proceso es igualmente realizado en JPEG, con la diferencia de que el tamaño de los rectángulos es configurable en JPEG2000, mientras que en JPEG es fijo. Cada uno de estos rectángulos es tratado independientemente como si fueran imágenes completamente separadas. Todas las operaciones siguientes son aplicadas de forma independiente a cada uno de los rectángulos.

Al separar y procesar la imagen por rectángulos, se reduce el uso de memoria y, debido a que los rectángulos son reconstruidos independientemente, se cuenta con la posibilidad de seleccionar cuáles rectángulos descomprimir en vez de reconstruir la imagen completa. También se puede comprimir cada zona con diferentes grados de calidad. Para realizar esto se debe especificar en los datos de entrada del algoritmo con qué calidad queremos comprimir cada zona. El siguiente paso es el más importante, en este se aplica la Transformada Wavelet.

Después de la trasformación, todos los coeficientes de trasformación obtenidos son cuantificados. Dicho proceso es similar al de JPEG, la diferencia de JPEG2000 es que no utiliza tablas de cuantificación, sino una función para obtener el valor entero por el cual serán divididos los coeficientes. La función es la siguiente: $q_b(u,v) = sign(a_b(u,v)) \times \left[\frac{|a_b(u,v)|}{\triangle_b}\right]$

Donde $q_b(u,v)$ es el resultado de cuantificación de cada coeficiente $a_b(u,v)$, b es el nivel de transformación en cuestión, u representa el ancho del nivel de transformación b, v representa el alto de la subbanda b y $\triangle_b$ es el valor de cuantificación para la subbanda b. Como mencionamos anteriormente, la calidad de la imagen

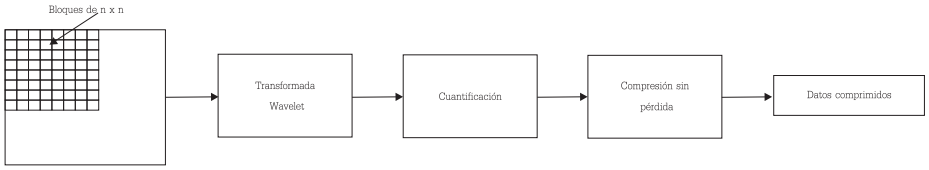


Figura 2.12: Esquema General del Algoritmo de Compresión JPEG2000.

requerida es un parámetro de entrada. Mientras la calidad en la reconstrucción de la imagen lo permita, la función se seguirá aplicando al siguiente nivel de transformación. En cada iteración $\triangle_b$ es ajustado hasta obtener la calidad requerida en los parámetros de entrada. Esta operación es con pérdida, para que sea sin pérdida el valor de cuantificación debe ser 1 y los coeficientes, enteros. JPEG2000 utiliza la Transformada Wavelet CDF 9/7 en el modo con compresión con pérdida. Para que los coeficientes sean enteros, JPEG2000 utiliza la Trasformada Wavelet integer 5/3 y evita pérdidas en la cuantificación. Como paso final, los coeficientes cuantificados son codificados por entropía. Este tipo de codificación asigna un código a cada símbolo del archivo por comprimir. Los códigos serán asignados dependiendo de la distribución de probabilidad de los mismos en el archivo por comprimir. Entre más probable sea cada símbolo de aparecer, más corto será el código que lo representa. Es importante recordar que cada uno de los procedimientos anteriores son realizados por cada rectángulo de la imagen. El esquema general del funcionamiento de JPEG2000, lo podemos observar en la Figura 2.12. En dicha figura se agrupan los pasos generales del compresor JPEG2000. El decodificador es el proceso inverso del codificador: primero se descomprime el resultado, posteriormente se aplica la cuantificación a la inversa y finalmente la transformada a la inversa.

2.2. Almacenamiento Digital de Imágenes

Una imagen digital $I(m, n)$ es un arreglo de dos dimensiones (m filas y n columnas). A cada celda de esta matriz $I(x, y)$ se le conoce como píxel. El tamaño de una imagen digital en escala de grises está dado por $m \times n \times p$ bits, donde p es la profundidad de color y 2^p es el número de colores que puede adquirir cada píxel. La imagen a color se representa digitalmente como tres arreglos bidimensionales (matriz) de $m \times n$. Cada matriz representa el resultado de una banda espectral RGB, por sus siglas en inglés de *red, green, blue* (rojo, verde, azul). El valor de cada píxel en una imagen en formato RGB está dado por la combinación de los valores de las coordenadas (x, y) de cada matriz [16]. Suponiendo que la imagen a color tenga las mismas dimensiones y profundidad de bits que la imagen I, entonces el tamaño en bytes de la imagen a color en espacio de color RGB es el tamaño en bytes de $I \times 3$.

Cabe mencionar que existen otros espacios de color además de RGB, sobre los cuales no profundizaremos en este documento. En el caso de formatos de imagen que permiten transparencia, se agrega otra matriz con las mismas dimensiones para la definición del valor alfa de cada píxel. Donde el valor alfa es el nivel de transparencia, Así, el valor de cada píxel estará dado por la combinación de cuatro valores en lugar de sólo tres. Las imágenes digitales pueden ser almacenadas en diferentes formatos. Algunos de los formatos de imagen más conocidos en la actualidad son los siguientes [16].

- **BMP**

  Bitmap (BMP) fue desarrollado por Microsoft e IBM de forma conjunta, es el formato propio del sistema operativo Windows [22], soporta hasta 16 millones de colores distintos. No se cuenta con la fecha exacta de creación de BMP debido a que surgió durante el desarrollo de Microsoft Windows; sin embargo, las primeras especificaciones no fueron formalmente publicadas sino hasta 1995 [9].

La estructura de un archivo BMP se compone de cuatro elementos principales: el encabezado general, el encabezado de la imagen, una tabla de color y los datos que representan la imagen. El encabezado general requiere 14 bytes para su representación. En este apartado del archivo BMP se especifican los datos necesarios para que cualquier aplicación identifique el tipo de formato, el tamaño del archivo y un apuntador al inicio de los datos que representan la imagen. El encabezado de la imagen tiene las especificaciones de la imagen, como ancho, alto, profundidad de bits, etcétera; el tamaño en bytes del encabezado es variable, puede ser mayor a 40 bytes.

Por otra parte, la tabla de color puede estar en tres formatos diferentes. Los primeros dos formatos son usados para asignar un valor en RGB a cada píxel, mediante un arreglo; esta configuración se puede utilizar cuando la profundidad de bits es 1, 4 u 8. Cada elemento del arreglo mide 2 bytes y representa un valor RGB. El tercer formato posible para la tabla de color se utiliza en caso de que la profundidad de bits sea de 16 o 32. En este caso se crea un arreglo de 3 dimensiones de enteros de 4 bytes, donde cada uno de los componentes representa las posibles combinaciones de tonos de rojo, verde y azul. Si la profundidad de bits es de 24, no existe paleta de colores dado que cada valor RGB del píxel es representado por 3 bytes. Para 16 y 32 bits de profundidad, también se puede omitir la paleta de colores y asignar directamente el valor RGB al píxel, siempre y cuando se establezca esta configuración en el encabezado de la imagen. Para el caso de 16 bits cada canal RGB se representa con 5 bits y el último bit es ignorado. Para el caso de 32 bits se utilizan 10 bits para cada canal RGB y los últimos 2 bits son ignorados.

El tamaño en bytes de la tabla de color está dado por $2^p \times a$, donde p es la profundidad de bits y a son los bytes de almacenamiento requeridos por cada elemento de la paleta de colores. Por último se agregan los datos de los píxeles.

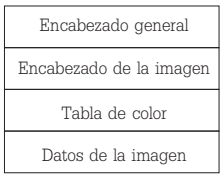


Figura 2.13: Estructura General de un Archivo BMP

Cuando se tiene paleta de colores, el tamaño en bits de los datos está dado por $m \times n \times z$, donde $m \times n$ son las dimensiones de la imagen y z son los bits requeridos para almacenar cada píxel; Si no se tiene paleta de colores, a siempre es fijo a 32 bits: 8 bits para cada canal RGB y los últimos 8 bits siempre son 0s representando un desperdicio innecesario de almacenamiento.

Como paso final, BMP permite aplicar compresión sin pérdida *Run-Length* a los datos que representan la imagen; aunque lo más común es que los archivos BMP no tengan compresión. En la Figura 2.13 se muestra la estructura general de un archivo BMP.

- **GIF**

  GIF por sus siglas en inglés de *Graphics Interchanche Format*, fue diseñado en 1987 por las empresas UNISYS y CompuServe. Las primeras especificaciones del formato GIF fueron llamadas GIF87a [10] y posteriormente CompuServe dio a conocer nuevas especificaciones respecto al formato de implementación (compatibles con las especificaciones anteriores) conocidas como GIF89a [15]. Sin embargo, la mayoría de los implementadores se quedaron con las primeras especificaciones. GIF permite animaciones y transparencia, aunque sólo permite hasta 256 colores distintos, lo que afecta la calidad de las imágenes con profundidad mayor a 8 bits por píxel y gran variabilidad en los tonos.

  La estructura de un archivo GIF se compone principalmente de cuatro elementos: el encabezado, una área para la descripción lógica del acomodo de los

componentes del archivo, una tabla de color global y, por último, los bloques que representan la información de cada imagen.

El encabezado de la imagen es el primer elemento del archivo GIF, su función es permitir a las aplicaciones identificar el archivo como GIF y determinar la versión de éste. Los primeros tres bytes identifican el tipo de formato y otros tres bytes determinan la versión, en total, seis bytes están reservados para el encabezado.

Una pieza importante del formato GIF es la tabla de color global. De acuerdo con la configuración, las imágenes individuales que pertenecen al archivo pueden usar una tabla de color global, o bien, definen una propia. Al establecer la tabla de color global para todas las imágenes, se reduce el tamaño de almacenamiento y, por ende, el manejo del archivo por parte del sistema es más fácil y rápido. Cada elemento de la tabla de color global está definido por tres bytes, uno para cada canal RGB; el tamaño de almacenamiento en bytes de la tabla de color global es $N \times 3$, donde N está limitado a 256. Lo cual significa que el formato solo permite 256 colores distintos.

Por lo que respecta al control del acomodo de múltiples imágenes en un solo formato, se define un área, denominada pantalla lógica, donde las imágenes individuales se mostrarán en el archivo GIF, la posición en donde las imágenes individuales se mostraran en la pantalla lógica se encuentra definida en la estructura que describe a cada imagen individual.

Para darnos una idea de cómo se visualizan las imágenes en la pantalla lógica, veamos la Figura 2.14: en este ejemplo tenemos tres imágenes que representan cada imagen individual, y el recuadro exterior simboliza la pantalla lógica. En esta sección también se especifican las dimensiones del área y el color del fondo (seleccionado de la tabla de color global). La pantalla lógica tiene reservados siete bytes para almacenamiento, de los cuales, cuatro son para determinar el

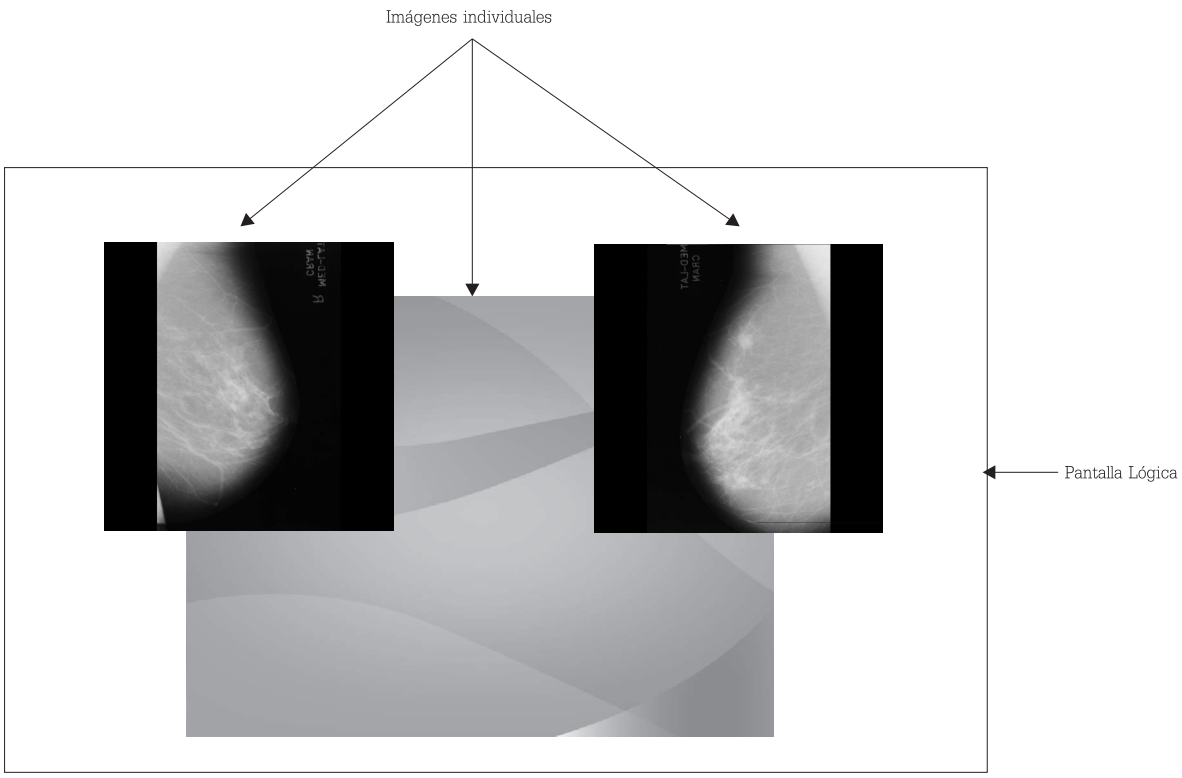


Figura 2.14: Ejemplo Gráfico del uso de la Pantalla Lógica con tres Imágenes.

ancho y el alto del descriptor de pantalla lógica, un byte es para el color del fondo y un byte se destina para el radio del píxel. El radio del píxel es una medida que indica la proporción del píxel entre el ancho y el alto de la imagen.

Los bloques de datos son el componente esencial del archivo GIF; estos bloques pueden ser de terminación de archivo, bloque de datos de imagen sin compresión o bloque de datos comprimidos de la imagen. El formato GIF define otros bloques adicionales que se pueden agregar al formato, pero, por cuestiones de enfoque, no los describiremos en esta sección. El tipo de bloque es identificado por el formato mediante los dos primeros bytes de cada uno. Los bloques que contienen los datos de la imagen poseen un encabezado con las especificaciones de las configuraciones de las imágenes, una paleta de colores local (en caso de que haya sido especificado de esa forma) y los datos comprimidos de la repre-

Figura 2.15: Estructura General de un Archivo GIF.

sentación de la imagen. El tamaño de cada bloque es variable y la cantidad de ellos también.

La estructura general de un archivo GIF se puede observar en la Figura 2.15. El formato GIF puede utilizar el algoritmo de compresión sin pérdida *Lempel-Ziv-Welch* (LZW) para comprimir la información de los valores que representan las imágenes [22].

- **TIFF**

  TIFF por sus siglas en inglés de *Tagged Image File Format* fue desarrollado en 1987 por la compañía Aldus y posteriormente adquirido por Adobe Systems. Este último publicó las especificaciones actuales del formato [4]. La principal particularidad del formato es que hace uso de etiquetas en el archivo para conocer las características de los datos que contiene el archivo; asimismo tiene la capacidad de almacenar más de una imagen en el mismo archivo. Este tipo de formato puede utilizar algunos algoritmos de compresión sin pérdida. TIFF puede soportar cualquier resolución de imagen y hasta 2^{32} colores distintos.

  La estructura principal de un archivo TIFF está compuesta por tres elementos: el encabezado de la imagen, los directorios de las imágenes y los datos que representan a éstas. Dicha estructura podemos observarla en la Figura 2.16. El encabezado de la imagen sirve para identificar el tipo de formato como tal,

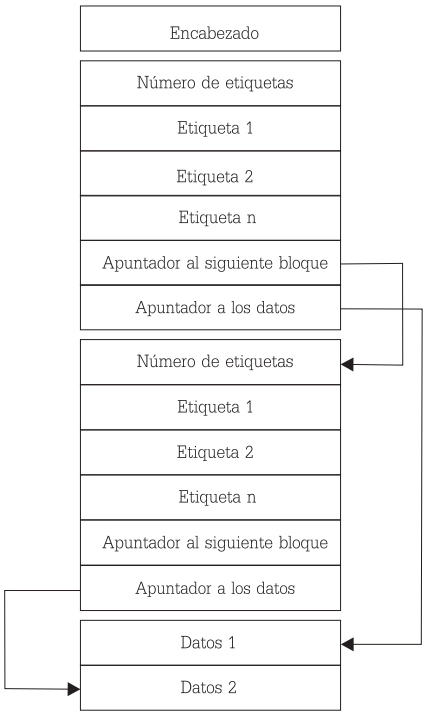


Figura 2.16: Estructura General de un Archivo TIFF.

indicar la versión y el apuntador al primer bloque de datos que representan la primera imagen y sus características. Para almacenar estos tres valores, el estándar TIFF asigna 8 bytes; el encabezado es el único bloque de datos de tamaño fijo, los demás bloques son de tamaño variable. Los siguientes bloques son los directorios de las imágenes, la parte donde se crea una estructura de datos para cada imagen, organización que contiene el identificador de la estructura, una lista variable de etiquetas (cada una de las cuales requiere 12 bytes de almacenamiento) que describen las características de la imagen, el apuntador hacia el siguiente bloque (en caso que lo haya) y la asociación con los datos de la imagen. El número de etiquetas es amplio, según el estándar existen 89 etiquetas disponibles para especificaciones [4]. En el último bloque se almacenan los valores de los datos que representan la imagen, esta información puede ser acomodada de dos formas: por tiras o por rectángulos. El acomodo por tiras

consiste en colocar, por secciones, n cantidad de filas consecutivas, donde n es un valor configurable. El acomodo por rectángulos consiste en dividir los datos de la imagen en rectángulos colindantes; el ancho y el alto de los rectángulos, así como la cantidad de ellos por imagen, son valores configurables.

Por último si así se configura, se comprimen los datos que representan la imagen. Existen varios métodos de compresión soportados por TIFF que se pueden aplicar a los rectángulos o tiras, según sea el caso. Algunos de estos algoritmos, son por ejemplo, codificación *Run-Length*, LZW y JPEG.

- **JFIF**

  JFIF por sus siglas en inglés de *JPEG File Interchange Format*, es un formato de archivo creado especialmente para el estándar de compresión JPEG (descrito anteriormente). JFIF apareció en 1992, fue propuesto por E. Hamilton [13] y aprobado por la empresa Microsystems. Los archivos JFIF tienen la extensión .jpg; de ahí que comúnmente se asuma que JFIF y JPEG en conjunto son el formato de imagen conocido como JPEG. Sin embargo, existen otros formatos de imagen que utilizan el estándar de compresión de imágenes JPEG [36]. JFIF es ampliamente utilizado en Internet para el intercambio de imágenes.

  JFIF emplea una serie de elementos a los que llama marcas, cuyo fin es identificar de qué se trata cada uno de los bloques de datos comprimidos y cómo tratarlos. Estos bloques son creados por el proceso previo de compresión, el cual puede tener variabilidad en el resultado y es lo que lleva a la necesidad de las marcas. Los bloques pueden contener ciertos tipos de marcas, todo depende del resultado de la compresión. Además de tener marcas para incrustar en los bloques de datos comprimidos, también se cuenta con marcas para especificar la información de los datos a las aplicaciones que leen los archivos. JFIF determina dos tipos generales de marcas: marcas aisladas y marcas con datos; las marcas aisladas miden dos bytes en tamaño, mientras que las marcas con datos miden cuatro

bytes (los últimos dos bytes de las marcas con datos indican la dirección de los datos que contienen).

Algunas de las marcas que se agregan al archivo son opcionales y otras no. La primera marca que se agrega es la que señala el inicio del archivo JFIF **SOI**, por sus siglas del inglés, *Start Of Image* y la última marca que se agrega es la que indica el final del archivo **EOI**, por sus siglas del inglés, *End Of Image*. El orden de las demás marcas es irrelevante para el formato sólo se tiene que considerar la restricción respecto al orden de la marcas: si la información de una marca se necesita en otra, la primera debe aparecer antes que la segunda. Además de las marcas de inicio y final, el archivo JFIF debe tener una marca de encabezado que indica especificaciones del formato. Estas especificaciones son: versión del formato, densidad horizontal y vertical de los píxeles, ancho y alto de las imágenes en miniatura (en caso de que las tuviera) y los datos que representan a las imágenes en miniatura. Esta estructura mide 14 bytes más el tamaño de las imágenes miniatura que son opcionales, se puede agregar una o más al formato, son imágenes comúnmente de menor dimensión que serán tratadas por el visualizador como imágenes complementarias de las demás imágenes. Posterior al encabezado, se agregan los resultados del compresor JPEG. El tamaño de los datos comprimidos es variable, depende de la imagen procesada. Finalmente, se agrega al archivo JFIF la marca de fin de archivo **EOI**. En la Figura 2.17 se expone la estructura general de un archivo JFIF.

- **PNG**

  PNG, por sus siglas en inglés de *Portable Network Graphics*, es un formato de imagen de mapa de bits creado para mejorar y reemplazar al formato de imagen GIF desarrollado por Boutell et al. en 1996 [3]. El formato de archivo PNG soporta hasta 16 millones de colores, mientras que el formato GIF sólo permite 256 colores; además PNG soporta transparencia. Para cada uno de los píxeles se

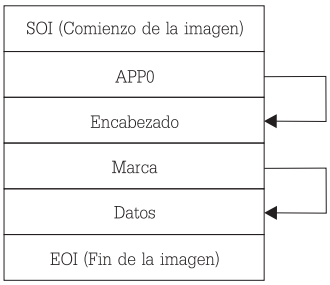


Figura 2.17: Estructura de General de un Archivo JPEG.

crea un valor paralelo que indica el grado de transparencia respecto al fondo, el factor transparente se incluye en la tabla de color; esta característica es conocida como canal alfa. Por su parte, GIF incluyó en las especificaciones GIF89A [15] la posibilidad de añadir transparencia a los colores, aunque está limitado a dos posibilidades: ser píxel transparente o no transparente, mientras que PNG permite 256 niveles de transparencia.

Otra de las características importantes de este formato es que utiliza un tipo de compresión sin pérdida y puede ser empleado cuando es esencial la reconstrucción idéntica de la imagen, a diferencia de JPEG, cuya compresión comúnmente es con pérdida. Es por dicha razón que JPEG y PNG son utilizados en aplicaciones con enfoques distintos. PNG comparte otra característica relevante con JPEG, que es la función de entrelazado, función que permite mostrar la imagen de forma gradual, mostrando primero una imagen con escasa calidad, y continuar mejorándola hasta llegar a la calidad máxima.

Las imágenes PNG usan la extensión de archivo .png y la mayoría de los exploradores y visores de imágenes actuales soportan dicho formato, que está conformado por un conjunto de bloques llamados *chunks*, una pieza imprescindible en el estándar. En la Figura 2.18, se puede visualizar el formato general de un *chunk* (bloque). Existen tres tipos de éstos: los primeros son los definidos por el estándar como obligatorios, cualquier programa que soporte archivos PNG

debe tener soporte para ellos; Los segundos son bloques de acceso público y pueden ser agregados por las aplicaciones; finalmente, se encuentra el tipo de bloques privados, definidos por los creadores de las aplicaciones. Estos últimos son propiedad del desarrollador y no serán reconocidos por aplicaciones sin previa adaptación especial.

La estructura del formato considera la posibilidad de que haya bloques que el decodificador no reconozca, y para esto establece un sistema de soporte ante bloques no reconocidos. Cada bloque es tomado como una unidad de procesamiento que tiene funciones definidas; algunos de ellos son indispensables y otros, opcionales. Los cuatro bloques indispensables definidos por el estándar PNG son: IHDR (encabezado de la imagen), con 13 bytes reservados para almacenamiento; PLTE (la paleta de color), con $2^n \times 3$ bytes para almacenamiento, donde n es el número de elementos de la paleta de colores, (las imágenes a escala de grises no tienen paleta de colores); IDAT (los datos de la imagen), por último, IEND (el marcador final del archivo PNG), con ocho bytes para almacenamiento. Estos mismos cuatro bloques son los que también tiene el formato PNG. La estructura general del archivo se puede visualizar en la Figura 2.19.

La estructura general de cada bloque en el formato PNG está compuesta por el tamaño en bytes de los datos, el nombre del bloque (con cuatro bytes reservados para esto), los datos propiamente y, por último, cuatro bytes reservados para el resultado de aplicar una función CRC, por sus siglas en inglés de *Cyclic Redundancy Check*, que funge como punto de partida para el descompresor. En caso que el descompresor obtenga un valor diferente al aplicar la misma función en los datos del archivo PNG, el descompresor podrá determinar que la imagen está corrupta.

PNG utiliza un formato de compresión basado en los algoritmos de compresión sin pérdida LZ77 y el algoritmo de Huffman para almacenar los píxeles de

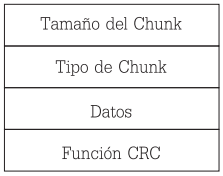


Figura 2.18: Estructura General de los Chunks.

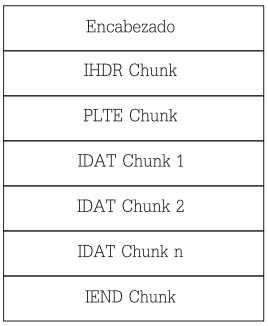


Figura 2.19: Estructura General de un Archivo PNG.

la imagen. Como contraparte a todas sus características favorables, PNG no soporta el almacenamiento de más de una imagen.

2.3. Compresión en Mastografías

Los algoritmos de compresión que se han aplicado a las imágenes de mastografías son, en su gran mayoría, del tipo reversible o sin pérdida de datos. Muchos trabajos comparan la eficiencia de estos algoritmos en las mastografías [17, 19, 23]. Aunque también se han realizado algunos estudios con algoritmos de compresión con pérdida [12, 25, 41]. Gran parte de dichos estudios evalúa la distorsión en las imágenes a medida que la Tasa de Compresión aumenta. Esta sección se enfoca en describir algoritmos utilizados en la compresión de mastografías digitales, que es un derivado de las imágenes médicas.

2.3.1. Compresión sin Pérdida

En esta sección describimos algunos trabajos en compresión de mastografías digitales relevantes en la literatura científica.

- Comparación de Algoritmos sin Pérdida en Mastografías Digitales
  Muchas de las investigaciones sobre compresión de imágenes médicas han comparado entre sí los algoritmos de compresión más relevantes en la literatura especializada. Un ejemplo de este tipo de trabajos es el de Khademi y Krishnan [18], quienes analizaron ocho algoritmos con el mismo conjunto de datos (imágenes de mastografías) para efectos de poder compararlos. La medida de eficiencia utilizada fue la Tasa de Compresión. De los algoritmos analizados, JPEG2000 obtuvo los mejores resultados en cuanto a Tasa de Compresión. JPEG2000 ha sido utilizado como referencia de comparación para las propuestas de compresión en imágenes médicas.

- Algoritmo de Compresión sin Pérdida para Mastografías Basado en Modelado por Contexto y Predicción Adaptativa
  Karimi y su equipo [17] realizaron una propuesta algorítmica de compresión para mastografías digitales en la que utilizaron la predicción adaptativa y el modelado por contexto. La predicción de un píxel es un proceso que consiste en asignar un valor estimado a un píxel tomando en cuenta los valores de sus píxeles vecinos. La selección de los píxeles vecinos y las operaciones aplicadas para determinar el posible valor del píxel en cuestión dependen de cada método de predicción. La predicción fue adaptativa porque para cada área se seleccionaba el método predictivo con la menor entropía. Los tres posibles predictores de píxel fueron: ALCM, por sus siglas en inglés de *Activity Level Classification Model* [32]; MED, por sus siglas en inglés de *Median Edge Detector* [37]; y GAP, por sus siglas en inglés de *Gradient-Adjusted Pre-dictor* [40]. Las áreas se determinaron

	v3	v4	v5	
v1	v2	x		

Figura 2.20: Vecinos Utilizados para Determinar el Valor de x en el Trabajo de Karimi y su Equipo [17].

mediante un proceso de segmentación. En este proceso se generó una máscara con tres valores posibles: fondo, tejido de la mama y tejido graso.

Además de utilizar la predicción adaptativa, Karimi y su equipo también emplearon el modelado por contexto. Este proceso consiste en aprovechar las características del objeto de estudio (para este caso, las mastografías digitales), al tomarlas en cuenta es posible mejorar el resultado de la predicción.

El algoritmo consta de seis etapas fundamentales: 1) preprocesamiento de la imagen, 2) creación de la máscara de segmentación, 3) identificación y predicción del píxel con base en sus vecinos, 4) selección del método de predicción por área, 5) extracción de las características del contexto, y 6) ajuste de la predicción mediante la aplicación de la regresión lineal, compresión de la máscara y compresión de los errores de la predicción.

En la etapa de preprocesamiento se comienza rotando la imagen 90 grados para aprovechar las áreas cercanas a 0 que pertenecen al fondo. Por la forma particular como se representa la mama en los estudios de mastografía, los renglones son interrumpidos más frecuentemente en zonas diferentes al fondo. Al girar la mama, los renglones se acomodan de una forma favorable para el predictor; También algunos renglones con valor 0 son eliminados para mantener el mismo tamaño en todas las imágenes.

Lo siguiente es obtener la máscara de segmentación, que es creada a partir de la generación de dos umbrales Con la utilización de los dos umbrales se genera una máscara con tres posibles áreas (fondo, tejido de la mama y tejido graso).

Después a cada área se le aplican los tres predictores y se genera la entropía del resultado de cada uno de ellos en cada área de la mastografía. Como resultado se selecciona por cada área el predictor que generó la menor entropía. El valor generado para cada píxel de la imagen está en función de sus vecinos. Para esto se deben identificar los vecinos de cada píxel. Se establecen cinco píxeles vecinos para generar el valor de cada píxel. Supongamos que x es el valor a predecir, en la Figura 2.20 se muestran los cinco píxeles vecinos considerados para dicha tarea.

El proceso posterior consiste en aprovechar las características de la mastografía para hacer una regresión lineal y ajustar los valores de la predicción respecto al valor original. Por cada píxel se obtienen ciertas características como, por ejemplo el valor del píxel en la máscara. Éste es el paso al que se le conoce como modelado por contexto. El objetivo es disminuir el error en la predicción de los valores.

Finalmente, se recalcula el error en la predicción de cada píxel respecto al original, y se comprime con la codificación aritmética [39]. Por último, se genera la salida del algoritmo. Dicha salida incluye la máscara comprimida con el algoritmo de compresión sin pérdida RLE, los errores en la predicción comprimidos por la compresión aritmética, el tipo de predicción por área y la fórmula de la regresión lineal.

- Algoritmo de Compresión sin Pérdida para Mastografías Basado en la Agrupación de Píxeles y Segmentación por Bloques Fijos
  Kumar et al. [19] también emplearon un método basado en predictores de píxel y segmentación por bloques; además aprovecharon las características particula-

v3	v2	v6	v7
v1	x		
v4			
v5			

Figura 2.21: Vecinos Utilizados para Determinar el Valor de x en el Trabajo de Kumar et al. [19].

res de las mastografías para mejorar la compresión. En el trabajo propuesto los bloques cuyos valores son 0s se tratan de forma diferente que los demás. Dichos bloques, los que son completamente 0s generalmente son los que pertenecen al fondo de la imagen de mastografía. A los bloques que no son completamente 0s se les aplica el método base que describiremos a continuación, para luego se realizar una técnica de agrupamiento de píxeles, y por último, se comprime con el algoritmo Huffman.

El primer paso que realizaron estos autores fue aplicar dos predictores. Uno de los predictores utiliza tres vecinos y el otro los 7 para determinar el valor del píxel en cuestión. En la Figura 2.21 se muestran cuales son los 7 píxeles vecinos utilizados para predecir el píxel x. El primer predictor se rige por la Fórmula 2.1 y el segundo predictor se rige por la Fórmula 2.2.

$$x = \begin{cases} min(v1, v2) & \text{si } v3 >= max(v1, v2) \\ max(v1, v2) & \text{si } v3 <= min(v1, v2) \\ v1 + v2 - v3 & \text{En cualquier otro caso} \end{cases} \tag{2.1}$$

$$x = v1 * 0.1 + v2 * 0.2 + v3 * 0.1 + v4 * 0.2 + v5 * 0.1 + v6 * 0.2 + v7 * 0.1 \tag{2.2}$$

Tras la aplicación de los predictores y la generación de una matriz de errores

para cada predictor, se dividen las mastografías en bloques de 4×4 píxeles. Los valores de las mastografías están altamente correlacionados, entonces, los bloques en sí tienden a tener valores similares. Para decidir cuál predictor usar para cada bloque, se obtiene la diferencia entre el valor máximo y el mínimo de cada bloque. El predictor que tenga la menor diferencia en cada bloque será el seleccionado para ese bloque en particular. Surge así el problema de cómo saber cuál de los dos predictores fue aplicado en cada bloque; esto es solucionado mediante un archivo binario, $b1$. Como sólo se cuenta con dos predictores, por cada bloque se asigna un bit en el archivo binario que representa cuál de los dos predictores es utilizado en cada bloque. De igual forma, se crea otro archivo binario, $b2$, para saber cuándo un bloque es de 0s completamente y cuándo no. Los bloques que son 0s únicamente son representados en el archivo binario con un bit y no se les aplica ningún procesamiento extra.

En el siguiente paso los valores máximo, mínimo y la diferencia d son obtenidos para cada uno de los bloques que no son completamente 0s. Luego se resta el valor mínimo a cada píxel; el número de bits requeridos para cada píxel está dado por $log_2(d + 1)$. Con cada bloque se agrega un encabezado, éste tiene el valor mínimo del bloque y los bits por píxel requeridos por bloque.

Después se trasforman los bloques de 4×4 en bloques de 2×2; para realizar dicha transformación se utilizan dos ecuaciones. La Ecuación 2.3 se aplica al bloque de 4×4 y la Ecuación 2.4 se aplica el resultado de la primera. Donde h es el valor máximo y para el caso de la Ecuación 2.3, i adquiere valores desde 1 hasta 4 y A es un arreglo bidimensional que contiene los valores del bloque. El resultado de aplicar la Ecuación 2.3 es un arreglo bidimensional de cuatro filas y dos columnas. El resultado final es un arreglo bidimensional de 2×2 y j adquiere valores desde 1 hasta 2.

$$\begin{aligned} x(i,j) &= A(i,j) * (h+1) + A(i,j+1) \text{ Si j=1} \\ x(i,j-1) &= A(i,j) * (h+1) + A(i,j+1) \text{ Si j=3} \end{aligned} \tag{2.3}$$

$$\begin{aligned} x(i,j) &= A(i,j) * (h+1) + A(i,j+1) \text{ Si j=1} \\ x(i,j-1) &= A(i,j) * (h+1) + A(i,j+1) \text{ Si j=3} \end{aligned} \tag{2.4}$$

La salida del algoritmo consta de: dos archivos binarios que son comprimidos primero con RLE y posteriormente con Huffman, los encabezados de cada bloque y los valores de los bloques comprimidos con Huffman.

2.3.2. Compresión con Pérdida

La compresión con pérdida en mastografías digitales ha sido evaluada recientemente. Empero, ya han surgido trabajos que nos dan una idea de qué tanto podría verse afectada la calidad diagnóstica de las imágenes reconstruidas con pérdida.

En 2000 Good et al. [12] evaluaron la detección de masas y agrupamiento de microcalcificaciones por un grupo de expertos. A dichos expertos se les solicitó evaluar 60 imágenes a las cuales se les habían aplicado diferentes escalas de compresión. Las escalas de compresión variaron desde imágenes sin comprimir hasta imágenes con una Tasa de Compresión de 101:1. La compresión utilizada fue un esquema de compresión de imágenes compatible con el estándar JPEG. A los mismos expertos, también se les solicitó evaluar de 0 a 100 la probabilidad de cada imagen de tener microcalcificaciones y/o masas. Se obtuvo como resultado que, estadísticamente, la compresión mayor a 70:1 mostró diferencias significativas en cuanto al diagnóstico emitido por el experto respecto a la imagen sin compresión.

En 2006 Pénedo et al. [25] realizaron un estudio donde utilizaron un algoritmo para la Detección de Microcalcificaciones (ADM) sobre las imágenes comprimidas con JPEG2000 con pérdida. Se evaluaron las variaciones en los resultados del ADM en seis

escalas de compresión diferentes: 1:1, 20:1, 40,1, 60.4:1, 80:1 y 106:1. Los experimentos se llevaron a cabo con 450 imágenes de mastografías digitales. Se evaluó el resultado de aplicar ADM en las imágenes comprimidas con JPEG200 con pérdida a diferentes Tasas de Compresión. Dicho resultado se comparó con el resultado de aplicar el ADM Las imágenes fueron tomadas de dos bases de datos privadas. Las imágenes estaban a 12 bits de profundidad y median 2000×2500 píxeles. Se concluyó en el estudio que el ADM utilizado detectó un porcentaje de microcalcificaciones similar en la imágenes sin compresión y en las imágenes con compresión hasta una tasa de compresión de 20:1. También se concluyó que el rendimiento del ADM en imágenes comprimidas con una tasa de compresión de hasta 40:1 es es aceptable respecto al análisis JAFROC por sus siglas en inglés de *Jackknife Free-Response Receiver Operating Characteristic*. Dicho análisis consiste en evaluar qué tanto conservan las imágenes comprimidas las características que los especialistas de la salud buscan como indicios de anomalías. [8].

2.3.3. Compresión Híbrida

Con la continua necesidad de hacer los métodos de compresión más eficientes, han surgido propuestas especializadas en mastografías digitales que combinan los métodos de compresión reversibles e irreversibles. Este tipo de propuestas se les conoce como algoritmos de compresión híbridos. Tayel y Mohsen [35] realizaron una propuesta híbrida donde extraen la zona de la mama y la comprimen con un algoritmo de compresión sin pérdida basado en diccionarios y el fondo lo asumen como negro total. Una propuesta similar a la de Tayer fue la de Xu et al. [41] en 2011 en donde utiliza compresión con pérdida en en área que no es relevante para el diagnóstico. Xu et al. [41] comenzaron por dividir la imagen de mastografía en zonas de interés con diferentes prioridades. Para esto, se creó una máscara de intensidad de grises en donde la prioridad va en descenso del valor más alto al valor más bajo, en este caso

el valor negro. Posteriormente cada una de las regiones de interés fue trasformada en regiones independientes a las cuales se les aplicó la Trasformada Wavelet integer-to-integer adaptativa, de acuerdo con el orden de prioridad etiquetado en la máscara. Por último, cada región de interés, con su respectiva prioridad, fue codificada con el algoritmo SPIHT modificado.

Xu et al. [41] probaron sus resultados en una imagen de mastografía digital. Su método con la Transformada Wavelet integer-to-integer 5/3 mostró una tasa de compresión 55.73 % mejor que JPEG2000 en la imagen analizada.

Capítulo 3
Propuesta algorítmica Híbrida

En este capítulo se describe detalladamente el algoritmo propuesto para la compresión de mastografías digitales. El algoritmo completo consta de dos partes fundamentales. La primera parte, que es propiamente el compresor, genera un archivo de tamaño reducido (con respecto a la imagen original). Este archivo contiene todos los elementos necesarios para la reconstrucción de la imagen de mastografía. La segunda parte del algoritmo se encarga de reconstruir la imagen de mastografía digital a partir del archivo creado por el algoritmo compresor. La integración del algoritmo compresor y el reconstructor forman nuestra propuesta algorítmica en compresión de imágenes de mastografías tipo híbrida.

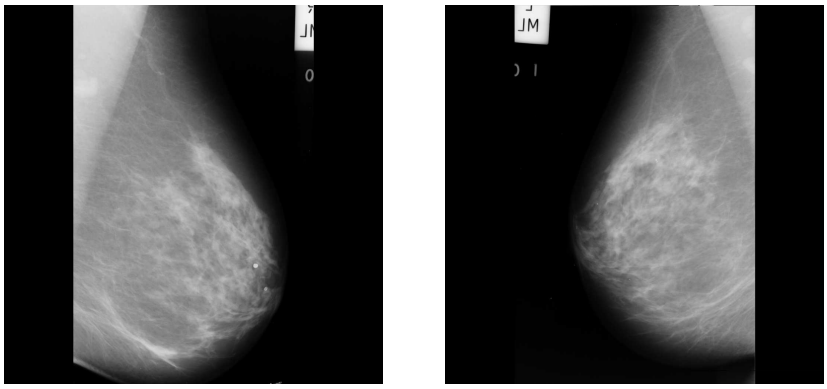
3.1. Enfoque Híbrido

En los últimos años han surgido propuestas algorítmicas en compresión que aprovechan las características particulares de las mastografías [17, 35, 41]. Los estudios de mastografías digitales son imágenes en escala de grises. La representación lógica de esta imagen es una matriz de $n \times m$. Cada celda de la matriz toma una valor 0 a $(2^b) - 1$, donde b es la profundidad de bits. Los tonos de grises que obtiene cada celda, construyen de forma abstracta la forma de la mama de la mujer. Entre más denso es el tejido de la mama el tono de gris es más claro. Pero, no todas las celdas de la matriz son parte de la mama, algunas de ellas, las que se encuentran en los extremos, generalmente son parte del fondo. Estas celdas, son de un color más obscuro que el de las celdas que pertenecen a la mama. Dichas celdas, las que pertenecen al fondo de la imagen de mastografía, no aportan información relevante para el diagnóstico.

Entonces la parte de la mama puede ser fácilmente identificable. Al separar la parte de la mama, nos quedamos con la región de interés o ROI (por sus siglas en inglés de *region of interest*); la región relevante para el diagnóstico. Los algoritmos híbridos en mastografías digitales trabajan de forma diferente la zona de interés y el resto de la imagen, utilizan técnicas de compresión sin pérdida para datos que representan la zona de interés y técnicas de compresión con pérdida en el resto de la imagen. Nuestra propuesta es un algoritmo de compresión tipo híbrido. Dicho algoritmo separa la parte de la imagen que representa la mama de la parte del fondo. Esta separación se obtiene mediante la utilización de un umbral de intensidad. Posteriormente se comprime la ROI (los píxeles que representan la mama) utilizando un algoritmo de compresión sin pérdida basado en diccionario. El resto de la imagen es considerado como negro total por el algoritmo descompresor. Para fines prácticos nombraremos a nuestra propuesta ACH (Algoritmo de Compresión Híbrido). A continuación describimos nuestra propuesta algorítmica en compresión de mastografías digitales.

- El algoritmo comienza por leer el archivo de computadora que representa la mastografía digital y lo convierte a su representación lógica como una matriz de $n \times m$. También se obtienen todos los parámetros de entrada del algoritmo. Dichos parámetros serán descritos en la Sección 3.2.

- En los parámetros de entrada se especifica la alineación de la imagen (izquierda o derecha) dependiendo si representa la mama izquierda o la mama derecha. En la Figura 3.1 mostramos los dos tipos de alineación posibles. ACH realiza una validación de la alineación de la imagen. En caso de que la imagen sea alineación derecha se obtiene el reflejo de la imagen de tal forma que visualmente parecerá el seno izquierdo. Esto con el fin de evitar realizar procesos diferentes dependiendo de la alineación de la imagen. Todos los procesos siguientes se aplican igualmente sobre la imágenes de cualquiera de las dos alineaciones. Al final, la alineación es enviada al descompresor como parámetro de entrada.



(a) Izquierda (b) Derecha

Figura 3.1: Alineación de las Mastografías.

- Una de las piezas fundamentales de nuestra propuesta es la segmentación de la imagen de mastografía. El resultado de la segmentación es guardado en otra imagen de referencia. Dicha imagen de referencia es llamada máscara. La segmentación separa la mama del fondo de la imagen.

(a) Original (b) segmentación (c) Depuración (d) Reconstrucción

Figura 3.2: Proceso de Compresión y Descompresión del Algoritmo ACH en una Imagen de Mastografía.

Para segmentar la imagen se utiliza un umbral. El umbral es un valor entero preestablecido en los parámetros de entrada. Cada uno de los píxeles en la imagen es comparado contra dicho umbral. El resultado de este paso es una matriz booleana de las mismas dimensiones que la imagen original. Un ejemplo de la segmentación de una imagen lo podemos observar el la Figura 3.2 en el inciso (b).

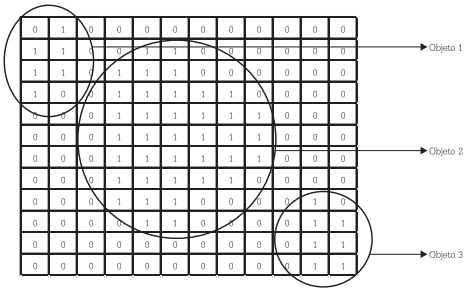


Figura 3.3: Ilustración de los Objetos Conectados Obtenidos Después de la Segmentación.

- Además de los píxeles pertenecientes a la mama, existen otros objetos en las mastografías con valores altos que pueden pasar el umbral de segmentación y aparecer en la máscara como valor verdadero. Para diferenciar estos objetos de la región de interés en la máscara solo basta con seleccionar el mayor área en cuanto a número de valores verdaderos y se descartan los demás.

  En la Figura 3.3 se muestra una matriz booleana que representa una imagen en donde se encuentran tres objetos. De los tres objetos claramente podemos observar que el objeto 2 es el que representa nuestra ROI debido a que es el que cuenta con la mayor cantidad de elementos. Lo que procede en esta situación es eliminar el objeto 1 y el objeto 3. Ahora nuestra máscara quedará como se observa en la Figura 3.2 (c).

- En algunas imágenes de mastografía el inicio de la ROI no se encuentra necesariamente adyacente al borde la imagen; la ROI puede presentar un desplazamiento. En la Figura 3.4 podemos observar a qué nos referimos con el desplazamiento. Para obtener el desplazamiento se hace una búsqueda del valor en la máscara desde el renglón medio hasta la primera columna con valor verdadero en la máscara.

- Como mencionamos anteriormente el objetivo de tener una máscara es conocer

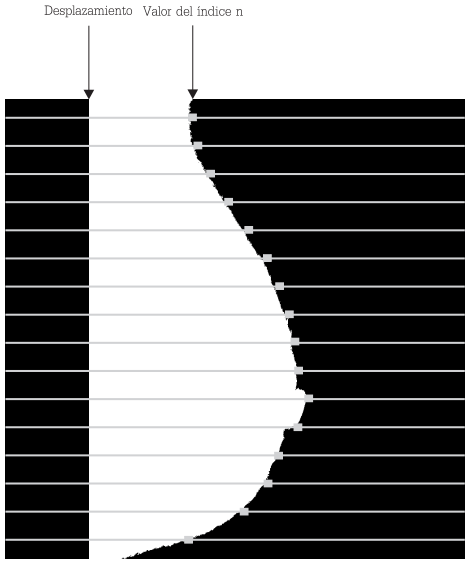


Figura 3.4: Desplazamiento e Índices de una Imagen de Mastografía Digital de Alineación Izquierda.

---

las medidas de la ROI y poder diferenciar la ROI del fondo. Una vez que contamos con el desplazamiento, nuestra propuesta prosigue a obtener los índices de la región de interés. Por cada renglón se obtiene un valor que representa el índice de la última columna perteneciente a la ROI del renglón en cuestión. La búsqueda de dicho valor se realiza desde el desplazamiento hasta el último valor verdadero en la máscara. Un ejemplo de algunos valores de estos índices se muestra en la Figura 3.4.

- Hasta este punto tenemos las dimensiones de la ROI por cada renglón. Con estas dimensiones obtenemos por cada renglón los valores de los píxeles que se encuentran desde el desplazamiento hasta el índice del renglón en cuestión.

- El penúltimo paso de nuestro algoritmo consiste en la construcción de una secuencia que contiene los datos necesarios para reconstruir nuestra imagen de mastografía. La estructura de dicha secuencia se puede observar en la Figura 3.5. El primer elemento de esta secuencia es el encabezado donde se incluyen

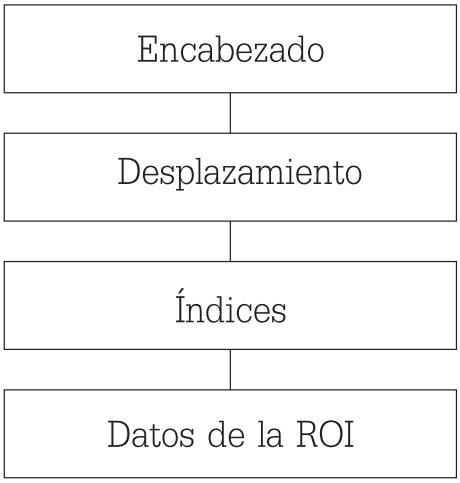
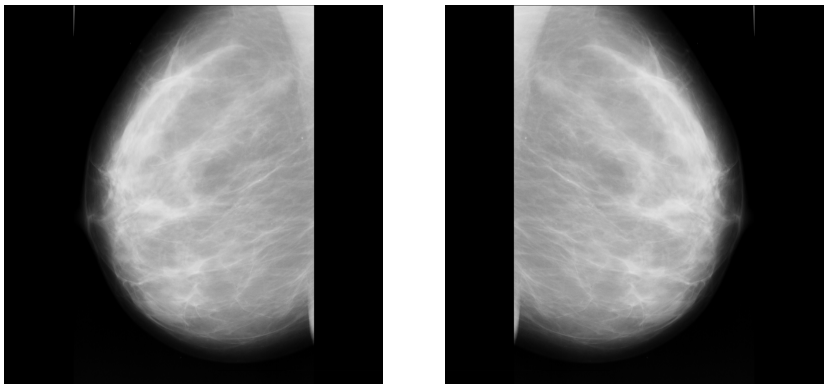


Figura 3.5: Estructura Creada por el Algoritmo Compresor ACH.

parámetros requeridos por el descompresor. Tales parámetros son: alineación de la imagen, ancho y alto de la imagen. Los siguientes elementos en la serie son el desplazamiento, los índices y los datos de la imagen.

- Por último se comprime la secuencia creada utilizando un algoritmo de compresión sin pérdida y se agrega, al principio de la secuencia comprimida información para identificar el algoritmo de compresión utilizado. La elección del algoritmo de compresión sin pérdida es adaptativa. Es decir, se elige un algoritmo de compresión por cada imagen dependiendo de el valor en TC (Tasa de Compresión) obtenido. El algoritmo que proporcione el mejor TC es el utilizado.

Lógicamente acabamos de describir el algoritmo propuesto, específicamente la parte de compresión. Por otra lado el algoritmo descompresor toma como entrada el archivo generado por el algoritmo compresor. Este archivo contiene todos los elementos necesarios para la reconstrucción de la mama. La reconstrucción de una imagen de mastografía realizada por el algoritmo de compresión la podemos observar en la Figura 3.2 (d).



(a) Alineación derecha (b) Mastografía Reflejada

Figura 3.6: Resultado de Reflejar una Imagen de Mastografía de Alineación Derecha.

3.2. Algoritmo para Comprimir Archivos de Mastografías

El algoritmo compresor toma como entrada una imagen de mastografía digital, la alineación de esta imagen (izquierda o derecha), el alto y ancho de la imagen y finalmente el umbral de segmentación. El alto y ancho de la imagen pueden ser obtenidos de los metadatos propios del formato de la imagen.

El umbral de segmentación es un valor previamente establecido para cada conjunto de imágenes. La salida del algoritmo compresor es una secuencia de datos comprimida utilizando un algoritmo de compresión sin pérdida. Dicha secuencia tiene un orden conocido por el algoritmo descompresor. La especificación de los parámetros de entrada y salida se puede observar en el Algoritmo 1 en las líneas 1-5. En las líneas 6-8 se encuentra una validación. Dicha validación verifica la alineación de la imagen. En caso de que la alineación sea a la derecha el algoritmo refleja la imagen de izquierda a derecha. Es decir, invierte el orden de los valores de los renglones. En la Figura 3.6 podemos observar el resultado de aplicar la función reflejar a una imagen de alineación derecha. De las líneas 9-12 se inicializan variables que posteriormente serán utilizadas en el Algoritmo.

Los dos ciclos anidados siguientes son diseñados específicamente para la creación de la máscara (líneas 13-21). Como mencionamos anteriormente la máscara es una imagen

Algoritmo 1 Compresor

Entrada:

$F1 \leftarrow Imagen\ de\ Mastografía\ a\ Procesar$
$Alineación \leftarrow Izquierda,\ Derecha$
$Umbral \leftarrow Umbral\ de\ Segmentación$

Salida: *Secuencia Comprimida*

$n \leftarrow F1.Alto\ de\ la\ Imagen$
$m \leftarrow F1.Ancho\ de\ la\ Imagen$
Si $Alineación = Derecha$ **Entonces**
  $Reflejar(F1)$
Fin Si
$ValorMedio = n/2$
$ElementoMayor \leftarrow 0$
$Máscara(n, m)$
$BW(n, m)$
Para $i \leftarrow 1, n$ **Hacer**
  Para $j \leftarrow 1, m$ **Hacer**
    Si $Máscara(i, j) > Umbral$ **Entonces**
      $Máscara(i, j) \leftarrow$ **Cierto**
    Si No
      $Máscara(i, j) \leftarrow$ **Falso**
    Fin Si
  Fin Para
Fin Para
$BW \leftarrow EtiquetarElementosConectados(Máscara)$
Para $i = 1$ hasta $i < Longitud(BW)$ **Hacer**
  Si $BW(i) > ElementoMayor$ **Entonces**
    $ElementoMayor \leftarrow BW(i)$
  Si No
    $EliminarBWenF2(Máscara, BW(i))$
  Fin Si
Fin Para
$Desplazamiento \leftarrow EncontrarDesplazamiento(ValorMedio, Alineación, n)$
$Índices \leftarrow ObtenerÍndices(Desplazamiento, Máscara, Alineación, n)$
$Datos \leftarrow ObtenerDatos(Desplazamiento, Alineación, m, n, Índices, F1)$
$Encabezado \leftarrow Alineación + n + m$
$Secuencia \leftarrow Encabezado + Desplazamiento + Índices + Datos$
$AC \leftarrow CAC(Secuencia)$
$SecuenciaComprimida \leftarrow AC + Comprimir(Secuencia, AC)$

binaria que nos indica las dimensiones de la ROI en la imagen. La máscara es creada a partir de una segmentación de los píxeles mediante un umbral. Cada uno de los píxeles de la imagen es comparado contra dicho umbral. Si el valor del píxel es mayor que dicho umbral, el valor correspondiente de la máscara se vuelve verdadero; de lo contrario dicho valor se mantiene en falso. Siguiendo el ejemplo utilizado en la Sección 3.1, la Figura 3.2 en el inciso (b) muestra el resultado de esta operación.

Una vez realizada la segmentación pudiera haber elementos que pasaron el umbral de segmentación pero que no son parte de la ROI. Sin embargo, de antemano sabemos que la ROI es el elemento de mayor área. Lo siguiente que realiza el algoritmo es encontrar los elementos conectados y descartar los que sean diferentes al más grande. Para esto se utiliza una función que agrupa los píxeles que están conectados entre si. Después de aplicar la función *EtiquetarElementosConectados* que se encuentra representada en la línea 22 se verifica qué tan grande son los elementos. BW es una matriz de tamaño $n \times m$ que contiene un identificador por cada grupo de píxeles en la máscara conectados entre sí. Las líneas posteriores representan la eliminación de los elementos en la máscara que son diferentes al más grande. Hasta este punto se tiene la máscara limpia. En la Figura 3.2 (b) se observa el resultado de la máscara limpia. Lo que sigue ahora es obtener el desplazamiento de la ROI en la máscara, los índices de cada renglón que delimitan la mama y los valores de los píxeles de la ROI.

Para obtener el desplazamiento de la ROI en la máscara, se cuenta con una función *EncontrarDesplazamiento* que recorre las columnas de izquierda a derecha desde el punto medio de los renglones hasta encontrar el comienzo de los valores verdaderos en las columnas de la máscara. En la Figura 3.4 se ilustra cual es el desplazamiento que se busca en las imágenes de mastografía. En la siguiente línea (31) se encuentra un vector llamado Índices que toma como entrada el resultado de la función *ObtenerÍndices*. Dicha función está definida en el Algoritmo 2.

El Algoritmo 2 toma como entrada el desplazamiento (mencionado anteriormente), la máscara, la alineación de la imagen y por último el ancho y alto de la imagen.

Algoritmo 2 ObtenerIndices

Entrada:
1: $Desplazamiento$
2: $Máscara$
3: $Alineación$
4: n
Salida: $Índices[n]$
5: **Para** $i \leftarrow 1, n$ **Hacer**
6: $cont \leftarrow Desplazamiento$
7: **Mientras** $Máscara(i, cont) = Verdadero$ **Hacer**
8: $cont = cont + 1$
9: **Fin Mientras**
10: $Índices(i) \leftarrow cont$
11: $cont \leftarrow n$
12: **Fin Para**

Como resultado, el algoritmo genera un vector de índices. Cada elemento del vector representa la terminación de la ROI en la máscara del renglón en cuestión. para obtener dicho vector se utilizan dos ciclos anidados, el primero recorre los renglones de la máscara y el segundo recorre las columnas partiendo desde el desplazamiento hasta encontrar un valor falso. El índice de la columna del último valor verdadero en la máscara por cada renglón de la imagen es lo que andamos buscando. En la línea 10 guardamos el valor buscado.

Regresando al Algoritmo 1, después de haber obtenido los índices el paso que sigue es obtener los valores del vector *Datos* (los valores de los píxeles de la ROI). Para esto se utiliza la función definida en la línea 32. Dicha función hace uso del Algoritmo 3. El Algoritmo 3 toma como entrada el desplazamiento, la alineación de la imagen, ancho y alto de la imagen, los índices previamente generados y la imagen original; la salida será una secuencia con los valores de los píxeles de la ROI. Para obtener dichos valores se utilizan dos ciclos anidados. El primer ciclo recorre los renglones de la imagen y el segundo recorre lo elementos pertenecientes a la ROI para el renglón en cuestión. Es decir, el algoritmo recorre y almacena en la serie de datos los valores de la imagen

Algoritmo 3 ObtenerDatos

Entrada:
1: $Desplazamiento$
2: $Imagen\ de\ entrada$
3: $Índices[n]$
4: n
Salida: $Datos[]$
5: **Para** $i \leftarrow 1, n$ **Hacer**
6: **Para** $j \leftarrow Desplazamiento, Índices(i)$ **Hacer**
7: $Datos \leftarrow Datos + Imagen\ de\ entrada(i, j)$
8: **Fin Para**
9: **Fin Para**

original que se encuentran entre el desplazamiento y valor del índice para el renglón en cuestión. En la línea 33 se crea el encabezado de la secuencia. Dicho encabezado tiene la alineación, el alto y el ancho de la imagen. Una vez que se tienen los datos necesarios para reconstruir la imagen lo que prosigue es colocar todos los datos de forma contigua. En este caso utilizamos la variable *Secuencia*. Esta variable tiene el encabezado de la imagen previamente construido, el desplazamiento, los índices y los datos de la mama.

Una vez construida la variable Secuencia nuestro algoritmo elige entre varios algoritmos de compresión sin pérdida. La función $CAC(Secuencia)$, que se encuentra en la línea 35, es utilizada para determinar cuál es el mejor algoritmo para comprimir la ROI de la imagen en cuestión. Dicha función toma como entrada una secuencia de datos y regresa un código representativo a alguno de los algoritmos de compresión utilizados. El resultado de la función está dado en función de la TC (Tasa de Compresión) más alta. Los algoritmos de compresión sin pérdida utilizados en el presenta trabajo se indican en el Capítulo 4.

Finalmente nuestro algoritmo comprime la secuencia con el algoritmo de compresión sin pérdida que seleccionó anteriormente. Al resultado generado, se agrega el valor representativo del algoritmo de compresión utilizado. La función *SecuenciaComprimida*

tiene el resultado de nuestro algoritmo compresor. A manera de ilustración el diagrama de flujo del algoritmo se encuentra en la Figura 3.7y la ilustración del algoritmo que crea la máscara de segmentación se encuentra en la Figura 3.8.

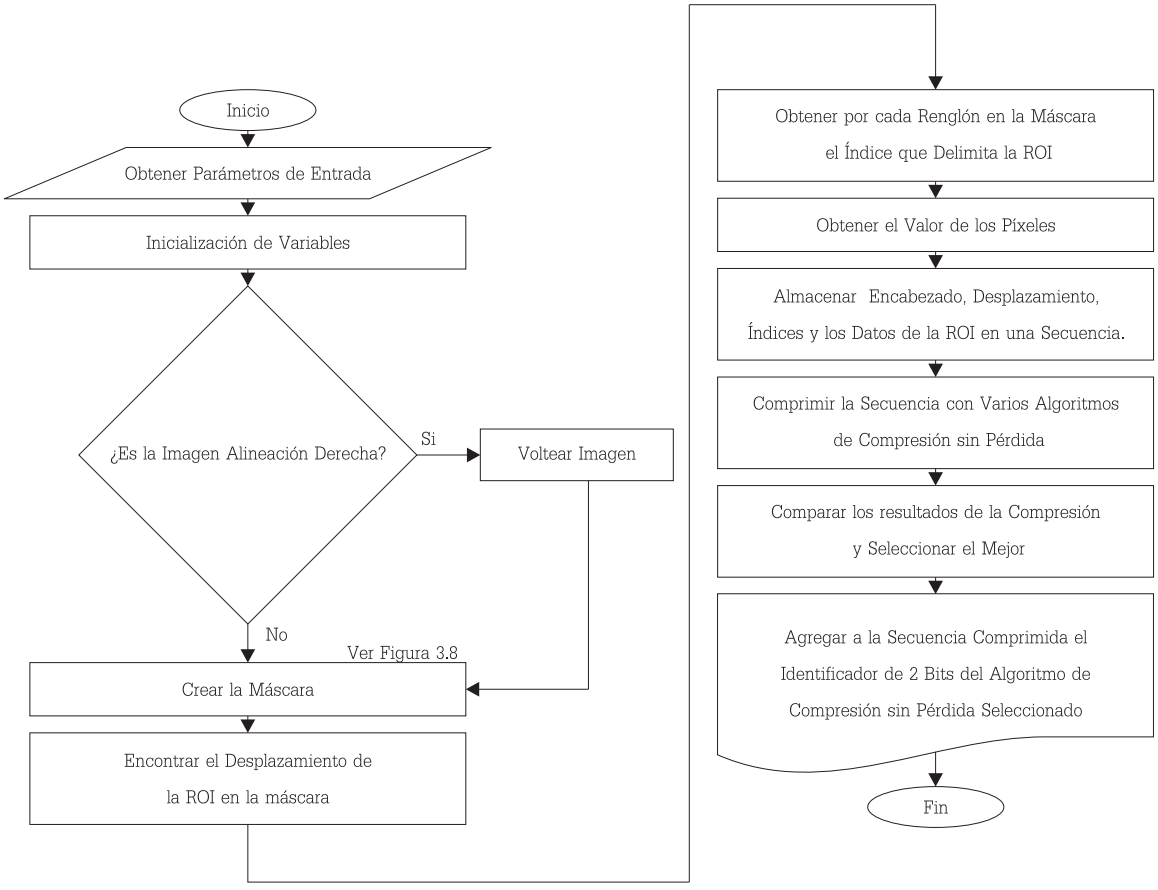


Figura 3.7: Diagrama del Algoritmo Compresor

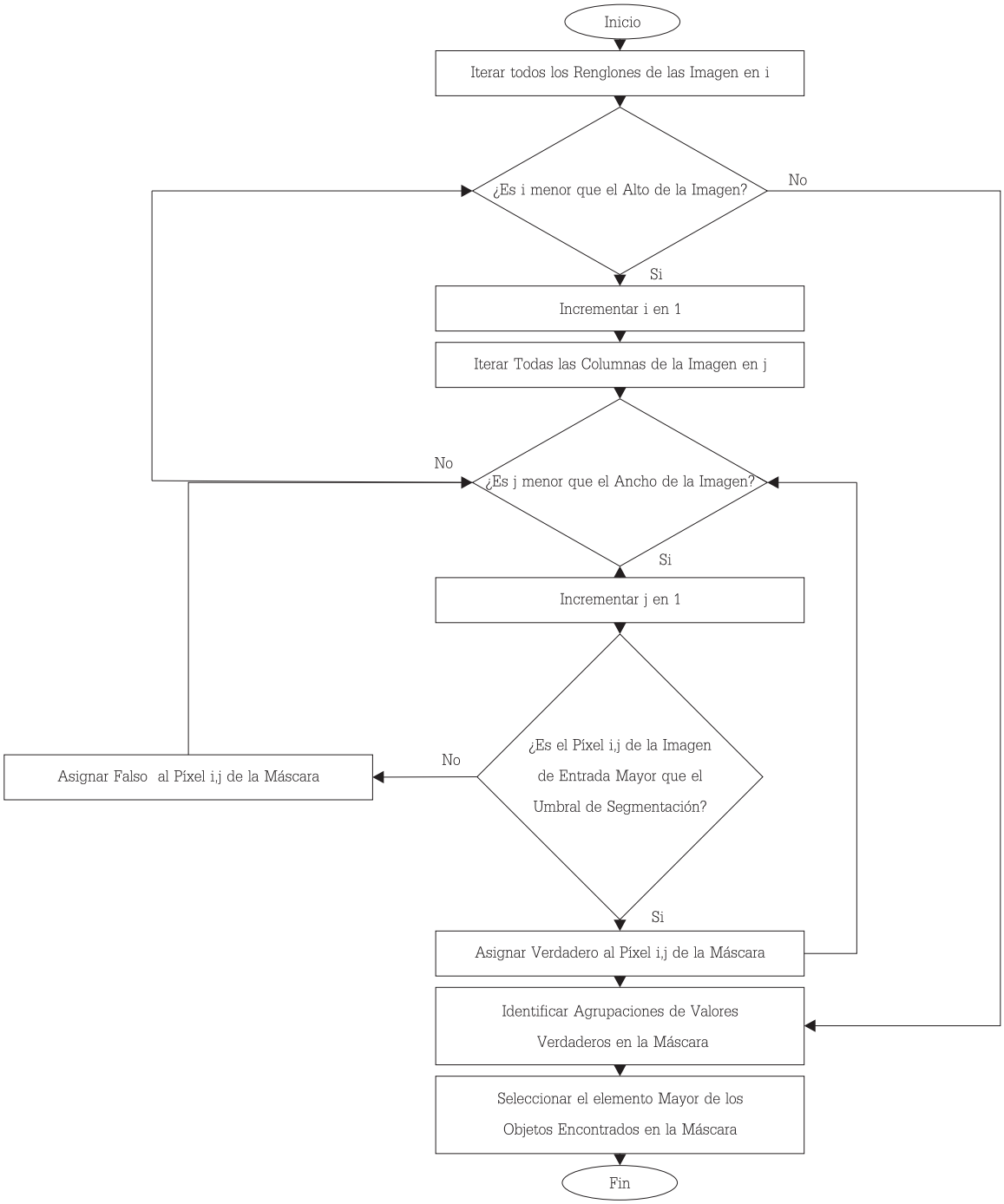


Figura 3.8: Diagrama del Algoritmo que Construye la Máscara que Delimita la ROI.

3.3. Descripción del Descompresor

El algoritmo descompresor toma como entrada la dirección completa del archivo electrónico creado previamente por el compresor. Dicho archivo debe ser el resultado del algoritmo compresor descrito anteriormente. Naturalmente la salida de nuestro algoritmo descompresor será una imagen de mastografía reconstruida.

Algoritmo 4 Descompresor

Entrada: *Archivo de entrada*
Salida: *Imagen reconstruida*
1: $AlgoritmoCompresor \leftarrow OAD(ArchivoEntrada)$
2: $a \leftarrow Descomprimir(ArchivoEntrada, AlgoritmoCompresor)$
3: $Alineación \leftarrow Archivo\ de\ entrada.Alineación$
4: $m \leftarrow Archivo\ de\ entrada.Ancho\ de\ la\ Imagen$
5: $n \leftarrow Archivo\ de\ entrada.Alto\ de\ la\ Imagen$
6: $Desplazamiento \leftarrow Archivo\ de\ entrada.Desplazamiento$
7: $Índices \leftarrow Archivo\ de\ entrada.Índices$
8: $Datos \leftarrow Archivo\ de\ entrada.Datos$
9: **Para** $i \leftarrow 1, n$ **Hacer**
10: **Para** $j \leftarrow Desplazamiento, Índices(i)$ **Hacer**
11: $Imagen\ reconstruida(i, j) \leftarrow Datos(cont)$
12: $cont \leftarrow cont + 1$
13: **Fin Para**
14: **Fin Para**
15: **Si** $Alineación = Derecha$ **Entonces**
16: $Reflejar(Imagen\ reconstruida)$
17: **Fin Si**

Lo primero que realiza el algoritmo reconstructor con la función OAD es revisar los primeros bits del archivo de entrada. Dependiendo de la combinación utilizada descomprime el archivo de entrada con el algoritmo de compresión correcto. Después de haber descomprimido el archivo lo que sigue es obtener los parámetros de entrada. Los parámetros se encuentran en el encabezado del archivo de entrada. Dichos parámetros son los siguientes: ancho y alto de la imagen, alineación de la imagen (izquierda o derecha). La asignación de los valores de los parámetros de entrada se realiza en el Algoritmo 4 en las líneas 1-8. El desplazamiento de la ROI, los índices y los valores de

los píxeles de la mama son obtenidos siguiendo la secuencia del archivo. Después de los parámetros de entrada el primer elemento que sigue es el desplazamiento, seguido por n (alto de la imagen) elementos que son los índices de la mama y finalmente los datos de la mama (los valores de los píxeles de la ROI). Siguiendo esta secuencia se asigna a las variables *Desplazamiento*, *Índices* y *Datos*, los valores correspondientes. Seguido de la asignación de las variables, el algoritmo prosigue a la ejecución de los dos ciclos anidados, mismos que se encuentran en las líneas 9-14 del Algoritmo 4. Dichos ciclos anidados son la principal funcionalidad del algoritmo descompresor. El primer ciclo recorre los renglones de la imagen y el segundo las columnas. Por cada columna de cada renglón se asigna el valor correspondiente. Los valores correspondientes se encuentran ordenados secuencialmente por renglones en la secuencia *Datos*. El recorrido de las columnas es desde el desplazamiento hasta el índice del renglón en cuestión. De la línea 15 a la 17 se verifica el valor de la variable *Alineación*. En caso de ser alineación derecha se aplica la función $Reflejar$. Esta función cambia el orden de las columnas en dirección horizontal. Así las imágenes que son alineación derecha regresan a su estado original. Con esta operación nos ahorramos leer de una forma los datos para las imágenes de alineación derecha y de otra forma las imágenes de alineación izquierda.

Finalmente tenemos la reconstrucción de la imagen de mastografía. Un ejemplo de una imagen normal y su reconstrucción se observa en la Figura 3.2. El diagrama de flujo del algoritmo Descompresor se ilustra en la Figura 3.9.

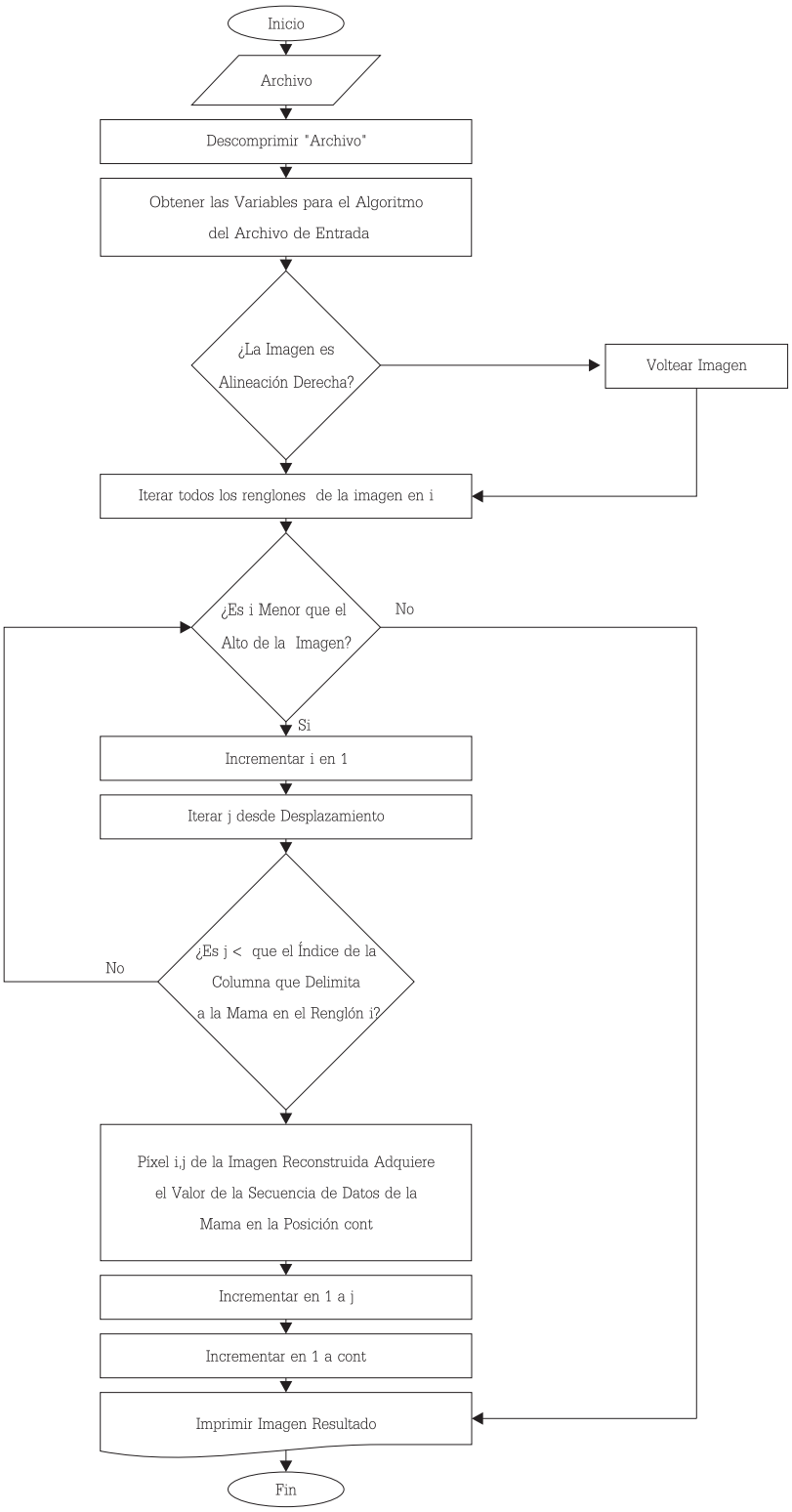


Figura 3.9: Diagrama del Algoritmo Descompresor

Capítulo 4
Evaluación Experimental

En este capítulo describimos los resultados de nuestra investigación. Para realizar dicha investigación se utilizó una base de datos de dominio público, la cual describimos detalladamente a continuación. Asimismo exponemos las herramientas utilizadas para la experimentación y su diseño, así como también los resultados (con apoyo en material visual).

4.1. Base de Datos

Para obtener los resultados de nuestra investigación se empleó una base de datos de mastografías digitales, la cual fue creada por la Organización de Análisis de Imágenes de Mastografías MIAS, por sus siglas en inglés de *Mammographic Image Analysis Society* [6]. Esta base de datos es de carácter público y ha sido utilizada a nivel internacional como punto de comparación entre estudios relacionados con mastografías. La base de datos está compuesta por mastografías, una por cada mama, tomadas a 161 mujeres del Reino Unido. Cada mastografía fue digitalizada a 50 micras por píxel con un Microdensitómetro Joyce-Loebl a 8 bits de profundidad; posteriormente las imágenes fueron reducidas a 200 micras por píxel y recortadas a un tamaño fijo de 1024×1024 píxeles.

La base de datos contiene en total 322 imágenes, de las cuales 208 son consideradas como normales (sin ningún indicio de malignidad) 63 benignas y 51 anormales (con lesiones malignas). Las mastografías pueden tener ciertos elementos que, según los expertos, indican sospechas de tumores malignos o benignos. Además de las imágenes de mastografía, la base de datos proporciona información adicional para todas las

imágenes, sobre el tipo de tejido que rodea la mama; puede ser graso, graso glandular o denso glandular. Las demás especificaciones son sólo para las imágenes que tengan algún tipo de anormalidad.

4.2. Configuración

Nuestra plataforma experimental cuenta con un procesador Intel Pentium con 2.40GHz de frecuencia, 4GB de memoria RAM y un disco duro de 500GB. El sistema operativo es Ubuntu 12.04, con lenguaje de programación MATLAB.

En nuestra propuesta algorítmica, detallada en el Capítulo 3 utilizamos compresión adaptativa. Para este trabajo en particular experimentamos con los algoritmos de compresión sin pérdida basados en diccionarios ZIP [21], RAR[11] y 7ZIP [24]. La elección del algoritmo de compresión sin pérdida se dio en función de su Tasa de Compresión (TC) para cada imagen. La TC es una medida de eficiencia que se explica a detalle en la Sección 3 de este Capítulo. Para identificar cuál de los tres algoritmos de compresión utilizamos en la compresión de cada imagen de mastografía, agregamos al inicio del archivo comprimido dos bits que representaban el algoritmo de compresión utilizado. A cada algoritmo se le fue asignado un identificador de dos bits. De esta forma realizamos la compresión adaptativa. El algoritmo reconstructor puede identificar cuál algoritmo de compresión fue utilizado previamente.

En la realización de los experimentos utilizamos un umbral de segmentación para la creación de la máscara. Dicho umbral fue obtenido del trabajo realizado por Tayel et. al [34]. En dicho trabajo los autores presentan un estudio estadístico de los valores pertenecientes a la mama en la base de datos MIAS. En el trabajo realizado por Tayer y su equipo se determinó que 32 es el mejor valor de umbral para segmentar la base de datos MIAS. Por dicha razón en nuestros experimentos utilizamos el umbral de segmentación con valor 32.

4.3. Medidas de Eficiencia

La medida más utilizada para medir la eficiencia en compresión es la Tasa de Compresión (TC). Esta medida se obtiene dividiendo el número de bytes utilizados para almacenar los datos originales entre el número de bytes necesarios para almacenar los datos después de haber aplicado el método de compresión [2].
Lo antes mencionado se describe también con la siguiente fórmula.

$$TC = \frac{Número\ de\ bits\ de\ los\ datos\ originales}{Número\ de\ bits\ después\ de\ comprimir} \tag{4.1}$$

Otra medida de eficiencia utilizada en el presente trabajo es la entropía de la información [28]. La entropía de determinada fuente de información nos proporciona la incertidumbre de ésta; de igual forma, se puede considerar como el grado de desorden de los datos, y depende directamente de la distribución de la probabilidad de los valores observados. La entropía de una fuente de información nos indica qué tan desordenados se encuentran los datos y, por ende, qué tan predecible es la información objeto de estudio. Desde otro punto de vista, la entropía depende directamente de la distribución de probabilidad de los símbolos. Entre más equiprobables sea cada uno de los símbolos de una fuente, más difícil será acertar cuál de los posibles símbolos se dará en cada instancia; por el contrario, si algunos valores tienen mucha más probabilidad de aparecer que otros, entonces la entropía de la imagen será baja [2].
En términos informáticos, la entropía nos da el número promedio de bits necesarios para almacenar una fuente de información, la cual puede ser una imagen, un texto o cualquier objeto que tenga representación electrónica. Así, en una imagen, por ejemplo, cada símbolo representa los posibles valores que puede adquirir cada píxel. La entropía de cada imagen es obtenida con la siguiente fórmula.

$$H = \sum_{i=1}^{m} p_i \; log_2 \frac{1}{p_i} \tag{4.2}$$

Donde p_i es la probabilidad para cada posible símbolo en la imagen y m es el número de posibles símbolos.

4.4. Resultados

En el Capítulo 3 se describió nuestra propuesta algorítmica de compresión híbrida. Dicha propuesta la hemos bautizado con el nombre de ACH por la abreviación de Algoritmo de Compresión Híbrido. Cuando hablamos de algoritmos de compresión híbridos nos estamos refiriendo a los algoritmos que implementan de forma conjunta técnicas de compresión sin pérdida y con pérdida. El algoritmo propuesto divide la imagen en dos secciones; la primera, es la parte del fondo de la imagen y la otra es la parte de la imagen que representa la mama, que es nuestra región de interés (ROI). Entonces la ROI contiene los valores de los píxeles que representan de forma abstracta la mama. ACH extrae de la ROI dos elementos fundamentales: un valor entero que representa el desplazamiento de la mama y los índices que indican la terminación de cada renglón de la mama. Posteriormente, ACH guarda en un archivo binario el desplazamiento, los índices y los valores de los píxeles de la mama. Por último, el archivo binario es comprimido con un algoritmo de compresión sin pérdida. A grandes rasgos acabamos de describir nuestra propuesta ACH.

Los elementos de los tres tipos de valores guardados en el archivo binario (desplazamiento, índices y los valores de los píxeles de la mama) requieren una cantidad de bytes para su almacenamiento. La cantidad de bytes requeridos por cada elemento esta determinada por la cantidad de valores distintos que cada elementos pueda adquirir. Para el caso particular de la base de datos MIAS, los bytes requeridos para almacenar cada uno de los elementos de los tres tipos de valores son los siguiente: 2

bytes para el desplazamiento, 2 bytes para cada índice y 1 byte para cada píxel de la mama (ROI).

Los experimentos de nuestra propuesta fueron realizados en todas las imágenes de la base de datos. Por cada imagen nuestro algoritmo genera un archivo binario. Dicho archivo cuenta con los elementos necesarios para la reconstrucción de la imagen de mastografía. El archivo binario es posteriormente comprimido con un algoritmo de compresión sin pérdida basado en diccionario. La elección del algoritmo de compresión es adaptativa, esto quiere decir que se elige de entre más de un algoritmo. Nuestra propuesta elige entre los algoritmos 7ZIP, RAR y ZIP. Ahora bien, pudimos seleccionar entre muchos otros algoritmos de compresión sin pérdida para comprimir el archivo binario. Pero decidimos utilizar 7ZIP, RAR y ZIP debido a que los tres son algoritmos de compresión basados en diccionario. Estos algoritmos, han demostrado ser, en general, una buena opción en compresión. Además, en el trabajo de [35], el cual es similar a nuestra propuesta, también experimentaron con algoritmos de compresión basados en diccionario.

El algoritmo de compresión es elegido en función de su valor en Tasa de Compresión. Esto quiere decir que entre más alta sea la Tasa de Compresión, mayor será la efectividad de dicho método de compresión. Dicha comparación se realizó con el objetivo de quedarnos con el mejor resultado en compresión para cada imagen. De los tres algoritmos evaluados, RAR fue el que mostró mejores resultados para todas las imágenes de la base de datos. Este se puede observar en el Anexo 1. El promedio en Tasa de Compresión de nuestra propuesta con cada uno de los algoritmos es de 5.55 para 7ZIP, 6.87 para RAR y 4.60 para ZIP. Nuestra propuesta con el algoritmo de compresión RAR superó a 7ZIP con una TC de 23 % superior y con una TC 49 % mejor que ZIP.

Ahora bien, tenemos los resultados de nuestra propuesta que en promedio tiene una Tasa de Compresión de 6.87. Se requiere un punto de comparación; para esto, utilizamos el algoritmo de compresión optimizado para imágenes JPEG2000 sin pérdida.

Como mencionamos anteriormente (en el Capítulo 2) JPEG2000 ha demostrado su buen rendimiento en compresión de imágenes médicas. Para esto realizamos la compresión de las imágenes de la base de datos con JPEG2000 sin pérdida y obtuvimos para cada una de ellas su Tasa de Compresión. Dichos resultados se pueden observar en la columna cinco del Cuadro 1 en el Anexo 1. Además, calculamos el promedio de Tasa de Compresión de JPEG2000 sin pérdida en las imágenes de la base de datos, siendo este de 5.98. Así nuestra propuesta presenta 14.88 % mejor Tasa de Compresión que JPEG2000 sin pérdida.

Otro de los experimentos que realizamos en el presente trabajo fue utilizar JPEG2000 sin pérdida en la ROI para analizar si JPEG2000 se ve favorecido por nuestro método de segmentación y así realizar una comparación entre métodos de compresión híbridos. La comparación se llevó a cabo entre nuestra propuesta y el el mismo método de segmentación de ACH con JPEG2000 sin pérdida en la zona de interés (los píxeles de la mama). Ahora bien, es necesario distribuir los píxeles de la ROI, de tal manera que puedan ser comprimidos por JPEG2000. En este momento tenemos representada la ROI en un vector de renglones donde la longitud de cada renglón es variable. Como JPEG2000 parte de un rectángulo, nos vimos en la necesidad de distribuir los renglones de la ROI en un rectángulo.

Para esto, se creó una imagen de 1 de alto por n de ancho, donde n es la cantidad de píxeles de la ROI. A esta imagen la llamamos i. Los valores de los píxeles de la mama fueron colocados en i de dos formas: la primera, fue renglón por renglón (VR), un renglón enseguida del otro de forma consecutiva hasta llegar a n; la segunda forma, los valores fueron acomodados por columna (VC), una columna tras otra de forma consecutiva hasta n. Lo siguiente fue aplicar JPEG2000 sin pérdida a cada imagen. Para que el algoritmo descompresor pueda reconstruir la imagen, además de los valores de los píxeles de la mama, los cuales se encuentran en la imagen, se requieren los índices donde los datos de la mama deben colocarse; datos que se dispusieron en un archivo y posteriormente se comprimieron con el algoritmo de compresión sin pérdida

RAR. Se utilizó RAR porque fue el que mejores resultados mostró en el proceso experimental realizado anteriormente.

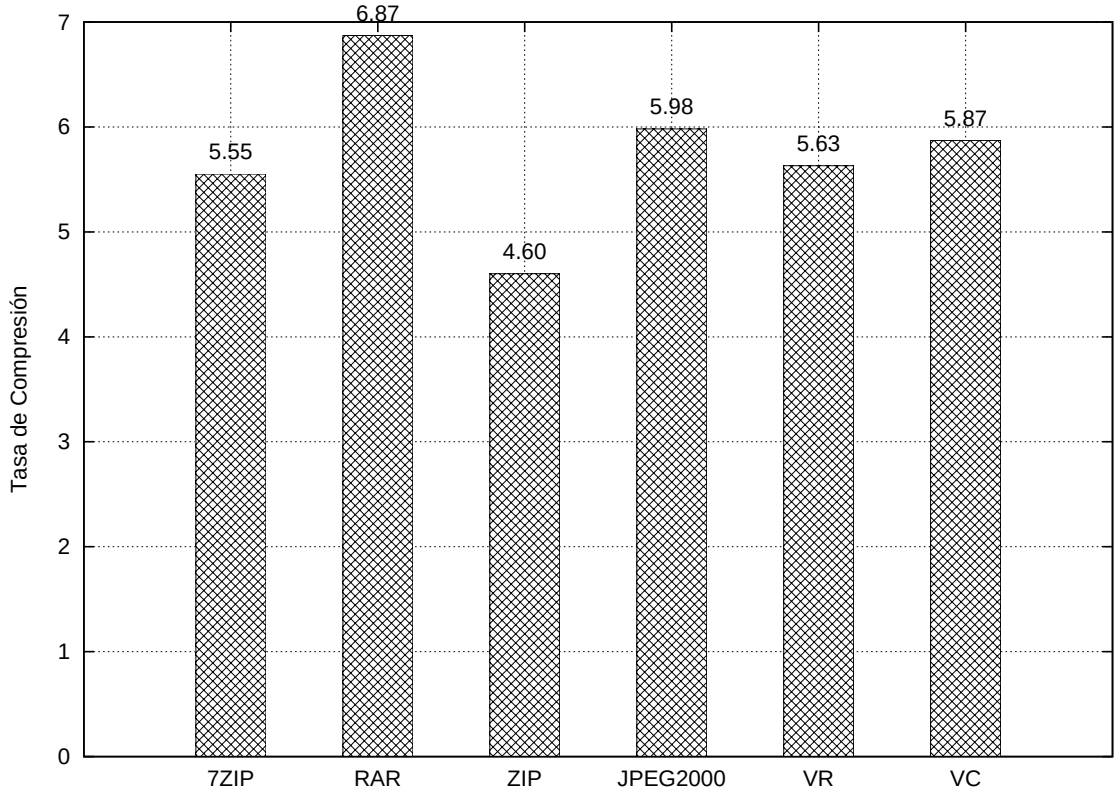


Figura 4.1: Promedio de TC, de Cada Uno de los Algoritmos Analizados en el Presente Trabajo, Utilizando Todas las Imágenes de la Base Datos.

Lo que prosigue en los experimentos es obtener la TC de JPEG2000 columnar (VC) y JPEG2000 por renglón (VR), en cada una de las imágenes de la base de datos MIAS. Para obtener la TC de estos dos experimentos se obtuvó el tamaño en bits de la imagen comprimida con JPEG2000 sin pérdida más, el archivo que contiene los datos de la ubicación de la ROI en la imagen. Dicho archivo está comprimido con el algoritmo de compresión sin pérdida RAR.

Los resultados de estos experimentos se pueden visualizar en la Figura 4.1, donde el experimento columnar está representado por la barra VC. La TC promedio para este experimento fue de 5.87. El experimento de aplicar JPEG2000 a los valores de la

mama acomodados por renglones se puede visualizar en la Figura 4.1, en la barra que lleva por nombre VR, con una TC de 5.63. VC mostró mejores resultados, que VR, aunque, con poca diferencia, tan sólo 4 % mejor. Sin embargo, el experimento VC no superó en TC a ACH. Nuestra propuesta fue mejor con un 17.03 %.

Hasta este punto tenemos que nuestra propuesta con el algoritmo de compresión RAR supera a JPEG2000 sin pérdida en imágenes de la base de datos MIAS sin procesar y también en imágenes procesadas con el mismo método de segmentación utilizado en ACH. Sin embargo, debemos preguntarnos si los algoritmos de compresión sin pérdida utilizados en la nuestra propuesta son mejores por si solos que JPEG2000 en la base de datos analizada. Para esto realizamos otro experimento donde analizamos los cuatro algoritmos (JPEG2000 sin pérdida, 7ZIP, RAR y ZIP) en imágenes sin procesar (directamente de la base de datos). Como resultado obtuvimos en promedio el algoritmo JPEG2000 sin pérdida obtuvo un promedio de TC de 5.44, mientras que RAR obtuvo 4.55. El algoritmo de compresión 7ZIP obtuvo 4.20; la TC más baja fue la generada por ZIP, con tan sólo 3.20. Estos resultados se pueden visualizar en la Figura 4.2. El algoritmo JPEG2000 sin pérdida superó en TC a RAR con 19 %, fue 29 % mejor que 7ZIP y 70 % mejor que ZIP.

Los experimentos anteriores nos ilustran que el método de segmentación utilizado en nuestra propuesta hace la diferencia sobre el resultado de compresión superior a JPEG2000. ¿Que pasaría si comparamos nuestra propuesta contra otro algoritmos de compresión que utilice el mismo método de segmentación?. HLC [35] es un algoritmo de compresión especializado para imágenes de mastografías similar a nuestra propuesta. El algoritmo de compresión HLC híbrido para mastografías utiliza el mismo método de segmentación [34] que emplea el algoritmo ACH. Tayel y su equipo retomaron para sus experimentos la base de datos MIAS, misma que usamos en este trabajo; aunque ellos sólo realizaron los experimentos con 25 imágenes de la base de datos. En el Cuadro 4.1 concentramos los resultados obtenidos por nuestra propuesta en las mismas imágenes con las cuales Tayel y su equipo trabajaron; además, agre-

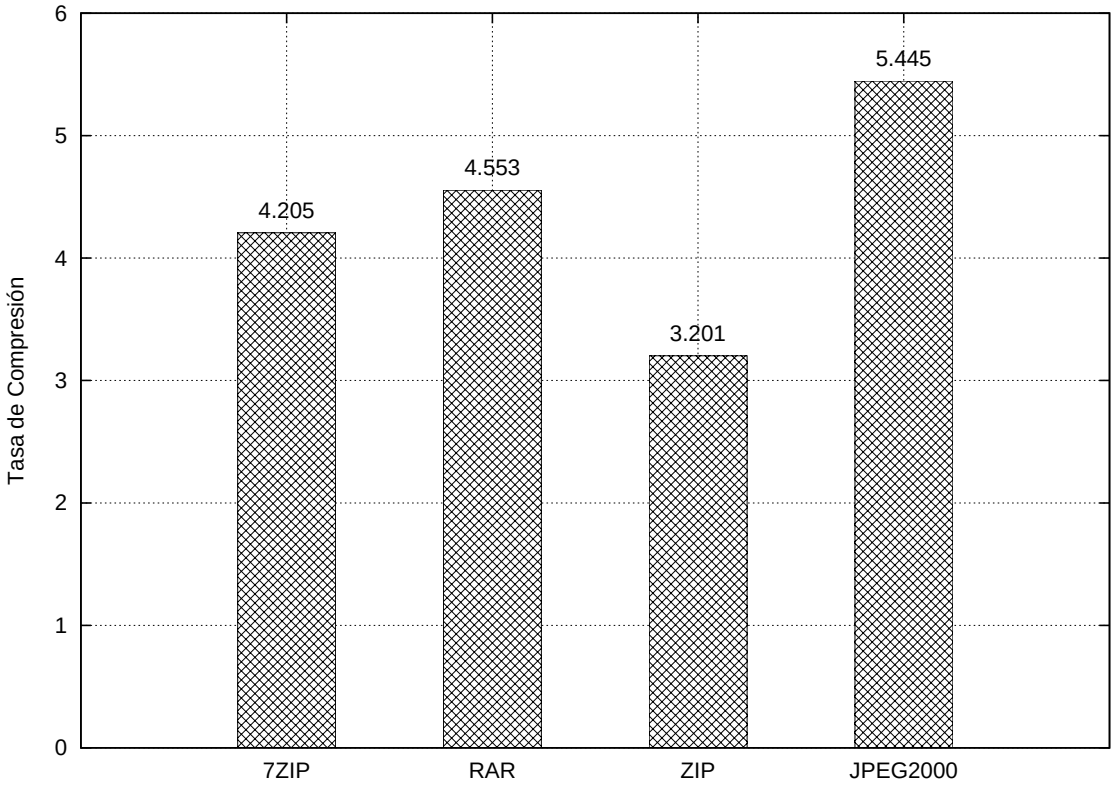


Figura 4.2: Resultados de los Algoritmos de Compresión sin Pérdida en Imágenes sin Procesar de la Base de Datos.

gamos los resultados de aplicar JPEG2000 sin pérdida a cada una de esas mismas 25 imágenes. Los datos expuestos en el Cuadro 4.1 hacen patente que en todos los casos ACH supera a HLC: en promedio, ACH aventaja a HLC con 17 % y a a JPEG2000 con 26 % para este conjunto de imágenes en particular.

Para finalizar este capítulo realizamos un análisis para verificar qué tan alejados están los resultados obtenidos de los algoritmos de compresión, del óptimo teórico (entropía). Para esto, obtuvimos por cada imagen su entropía y el resultado en bits por píxel de la compresión. Los algoritmos de compresión analizados fueron: L7ZIP, RAR, ZIP y JPEG2000 sin pérdida. En la Figura 4.3 se puede visualizar la relación entre el resultado de los algoritmos de compresión en bits por píxel con la entropía asociada de la imagen antes de comprimir. En dicha figura se puede observar que

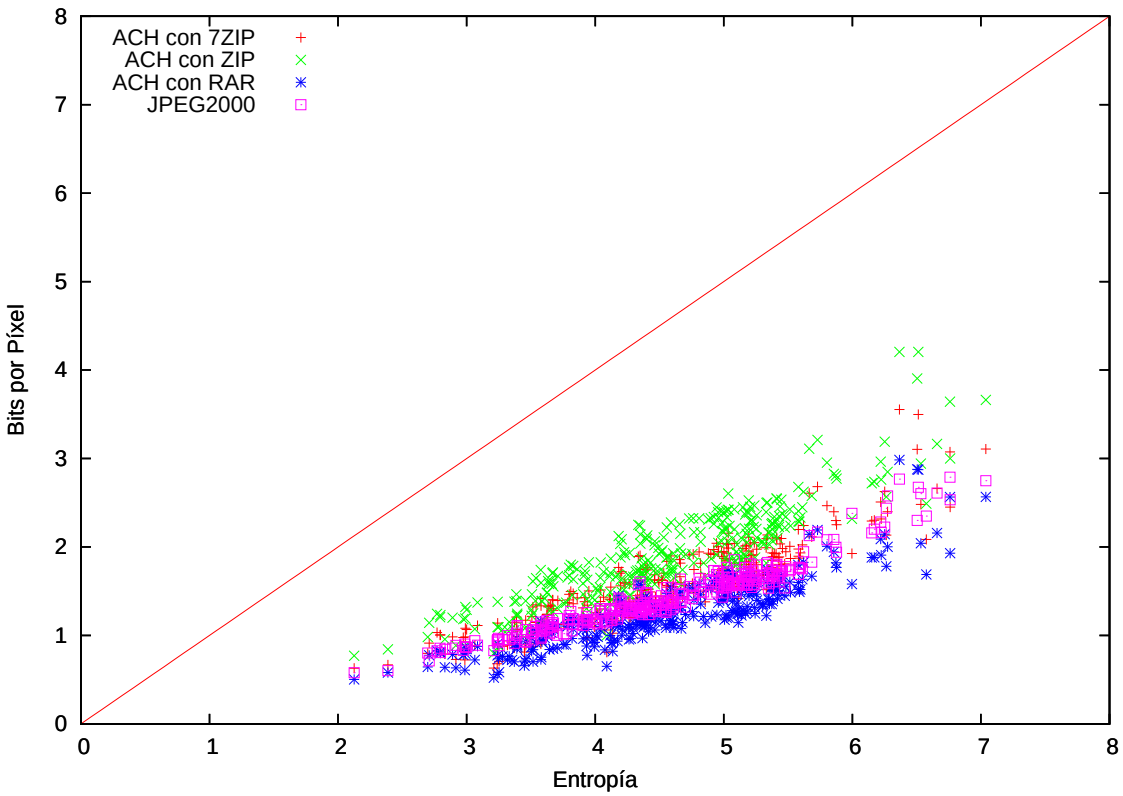


Figura 4.3: Resultados en Bits por Píxel de JPEG2000, ACH con 7ZIP, ACH con ZIP y ACH con RAR, ordenados por Entropía.

todos los algoritmos de compresión analizados requieren menos bits por píxel que el óptimo teórico. Entonces, debemos explicar cómo es posible que se presente este fenómeno.

La entropía, como mencionamos anteriormente, es una medida de eficiencia que nos proporciona la cantidad mínima de bits por píxel requeridos (el óptimo) para almacenar una fuente de información (en nuestro caso una imagen). Sin embargo, el resultado de nuestros algoritmos proporciona un número más bajo que el de la entropía asociada para cada imagen. Se pudiera creer que esto sucede debido a que nuestra propuesta con 7ZIP, RAR y con ZIP en realidad son algoritmos de compresión con pérdida. Pero en el caso de JPEG2000 se trata de un algoritmo de compresión sin pérdida y aun así, para todas las imágenes JPEG2000 requiere una cantidad menor de bits por

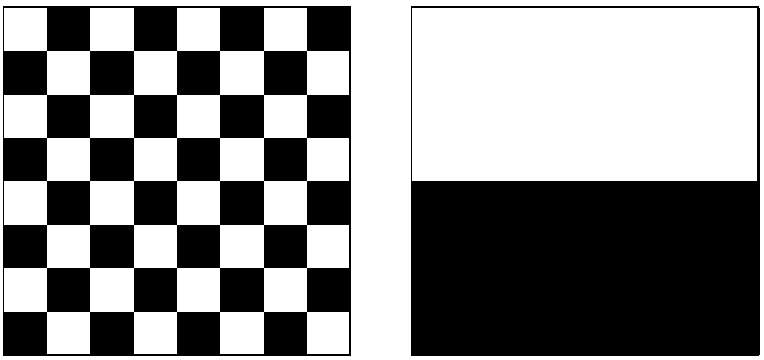


Figura 4.4: Imágenes con la Misma Cantidad de Píxeles blancos y Negros Acomodados de Diferente Forma pero Tienen la Misma Entropía

píxel, por debajo del óptimo según la teoría de la información. Esta es una de las partes más interesantes de los algoritmos de compresión; para entender mejor esta situación veamos un ejemplo. En la Figura 4.4 podemos observar dos imágenes; ambas de las mismas dimensiones (8×8) y con la misma cantidad de píxeles blancos y negros; aunque, los píxeles se encuentran distribuidos de forma diferente en cada una de las imágenes; si obtenemos para cada una de ellas su entropía, tenemos que para ambas imágenes la entropía es de 1. Ahora bien, por otro lado aplicamos un método de compresión (RLE solo como ejemplo) a cada imagen y obtenemos que para la imagen del inciso (a) en la Figura 4.4 se requieren 9 bits por píxel y para la imagen del inciso (b) se requiere 0.5625 bits por píxel.

Es así como los algoritmos de compresión usan la redundancia de los píxeles para reducir la cantidad de bits por píxel requeridos. Sin embargo, esto sucede debido a que los valores de los píxeles están correlacionados; en casos donde no lo están la compresión en lugar de disminuir los bits por píxel requeridos los aumenta. De ahí la importancia de seleccionar un método de compresión adecuado para cada tipo de imagen.

Cuadro 4.1: Resultados en TC para 25 Imágenes de la Base de Datos Utilizadas en el Estudio de Tayel y su Equipo.

Imagen	JPEG2000 sin pérdida	HLC	ACH
1	7.08	7.21	8.12
5	4.73	4.42	8.79
16	6.71	6.24	7.23
17	8.50	10.38	11.05
18	8.26	9.71	10.00
19	4.70	4.01	10.87
22	6.08	6.43	7.30
25	4.49	4.08	4.74
27	4.74	4.75	5.25
30	6.08	6.27	7.13
33	8.40	10.65	12.15
34	8.14	9.29	10.60
35	8.42	10.51	11.35
36	8.25	9.86	10.99
37	8.41	9.27	10.43
38	9.27	8.93	10.08
39	9.98	10.90	12.39
40	13.18	14.01	15.89
42	5.94	5.99	6.74
43	8.18	9.69	10.98
44	7.73	8.41	9.54
45	6.09	6.46	6.93
46	5.87	5.89	6.82
47	6.48	7.22	8.24
49	5.10	4.53	5.12
Promedio	7.23	7.80	9.14

Capítulo 5
Análisis de Resultados

En el presente trabajo se presenta una nueva propuesta para la compresión de imágenes de mastografías digitales. Se propone un algoritmo de tipo híbrido (llamado ACH), es decir, una clase de algoritmo que combina la compresión sin pérdida y la compresión con pérdida. La compresión sin pérdida se aplica en la zona de interés (ROI) y la compresión con pérdida se aplica en la parte de la mastografía que es considerada como el fondo de la imagen: para determinar la ROI, que en esencia consiste en los píxeles de la mama, se realiza un proceso de segmentación de la imagen; además de la ROI, el algoritmo reconstructor necesita conocer la posición en la imagen donde se colocará cada uno de los renglones que representan la mama, valores que son también agregados a un archivo junto con los datos de la mama (ROI); finalmente, el archivo es comprimido con un algoritmo de compresión sin pérdida.

Cabe destacar que la elección del algoritmo de compresión sin pérdida es adaptativa, esto es: por cada imagen se elige uno de entre tres algoritmos de compresión sin pérdida basados en diccionario, elección que está en función de su rendimiento, medido por su Tasa de Compresión (TC). Es preciso señalar que, por cuestiones de delimitación de alcance, en la presente propuesta sólo utilizamos tres algoritmos, aunque ACH está preparado para comparar las TC de cualquier cantidad de algoritmos de compresión. Así, para este trabajo en particular, los algoritmos utilizados fueron: 7ZIP, RAR y ZIP.

Para comprobar la efectividad de nuestro algoritmo de compresión propuesto, realizamos una serie de experimentos, cuya descripción se detalla en el Capítulo 4. Durante tal proceso de experimentación, comparamos nuestra propuesta con el bien conocido algoritmo de compresión JPEG2000 sin pérdida, y además comparamos ACH con un

algoritmo de compresión híbrido similar a éste. A fin de obtener resultados fiables y contundentes, realizamos varios experimentos.

El primero de nuestros experimentos consistió en analizar la TC de los algoritmos 7ZIP, RAR y ZIP en cada una de las imágenes de la base de datos pública MIAS; esto, con el propósito de quedarnos con el mejor de los algoritmos de compresión para cada imagen. Con base en los resultados, podemos decir que: 1) el algoritmo de compresión sin pérdida RAR proporcionó los mejores resultados en cuanto a TC, para todas las imágenes de la base de datos; 2) ACH con RAR fue mejor, en cuanto a TC, que ACH con 7ZIP y ACH con ZIP (24 % y 49 % respectivamente). De ahí que nuestra propuesta utilice para todos los casos el algoritmo de compresión sin pérdida RAR.

Luego de analizar los resultados de ACH con los tres algoritmos de compresión sin pérdida, lo comparamos con JPEG2000, un algoritmo de compresión sin pérdida optimizado para imágenes que ha demostrado su efectividad en imágenes médicas. Es precisamente por esta razón que, así en la literatura científica como en este trabajo, es utilizado como punto de comparación para nuevas propuestas en el área de compresión de imágenes médicas. Los resultados obtenidos nos indican que nuestra propuesta superó a JPEG2000 sin pérdida: obtuvo, en promedio, una TC 15 % mayor.

Después, siguiendo con las pruebas de JPEG2000, realizamos otros dos experimentos, que consistieron en acomodar los valores que representan la ROI en una imagen de $1 \times n$. Dicho acomodo se llevó a cabo de dos formas: la primera por renglones y la segunda por columnas. Sin embargo, ninguno de los dos métodos de compresión superó a JPEG2000 sin pérdida en las imágenes tomadas directamente de la base de datos, debido a que el método de compresión utilizado por JPEG2000 está optimizado para señales de dos dimensiones y nosotros convertimos los valores de los píxeles en una señal unidimensional.

En adición, nos interesaba comparar los resultados de los algoritmos de compresión sin pérdida utilizados en nuestra propuesta (7ZIP, RAR, ZIP) con JPEG2000 en

las imágenes sin procesar de la base de datos, sin aplicar a cada imagen el proceso previo de segmentación utilizado en ACH. Lo anterior, con el fin de conocer qué tan beneficioso resulta nuestro método de segmentación. Al realizar tales comparaciones obtuvimos que, en promedio, JPEG2000 sin pérdida es mejor que todos los demás algoritmos, incluso mejor que RAR, al que supera con 19.56 % en TC.

La supremacía de JPEG2000 se desprende del hecho de que es un algoritmo de compresión optimizado para imágenes, mientras que los demás algoritmos trabajan a nivel de bytes, sin aprovechar la redundancia entre los valores de los píxeles. Así y todo, nuestro método de segmentación ayuda a mejorar los resultados de RAR, un algoritmo cuyo campo de especialización se limita a la compresión de datos en general; en promedio, ACH aporta a RAR una TC 24 % mayor que la alcanzada por JPEG2000. De frente a tales resultados, podemos decir que utilizar nuestro método de segmentación en imágenes de mastografía puede reducir casi 7 veces el espacio de almacenamiento de la imagen original.

Es importante recordar que las imágenes de la base de datos MIAS retomadas en el presente trabajo para efectuar nuestra experimentación ya se encontraban recortadas, gran parte del fondo fue eliminado previamente. De lo contrario, el rendimiento de ACH hubiera sido superior, pues ACH sólo necesita guardar los datos de la ROI y la ubicación de ésta en la imagen, sin importar el tamaño del fondo, en tanto que JPEG2000 no implementa tal discriminación y, por ende, para él la imagen por comprimir es siempre más grande. Esto nos indica cuán importante y provechosa es la segmentación utilizada como parte de la presente propuesta, pues si los significativos resultados aquí expuestos fueron obtenidos en un ambiente controlado, al ejecutar ACH en mastografías de tamaño real la TC sería mucho mayor.

Como parte de este estudio también comparamos nuestra propuesta con Hybrid Loosless Compression (HLC), un algoritmo de compresión híbrido similar a ACH, uno que utiliza el mismo método de segmentación, y nuestra propuesta lo superó con 17 % en TC. HLC realiza el proceso de segmentación y convierte los valores que no son parte

de la ROI en ceros; a la imagen resultante, la fusión de los valores del fondo convertidos en ceros y los datos de la ROI, se le aplica posteriormente el operador lógico XOR; finalmente, la imagen es comprimida con un algoritmo de compresión sin pérdida basado en diccionario. HLC comprime todos los valores del fondo, aunque estos son negros totales y resultan irrelevantes para el diagnóstico, mientras que nuestra propuesta sólo almacena la posición de la ROI en la imagen, lo que implica que ACH trabaja con un menor número de valores para comprimir.

Para finalizar nuestro diseño experimental, analizamos la relación entre la entropía de cada imagen con los bits de almacenamiento por píxel requeridos de ACH con 7ZIP, ACH con RAR, ACH con 7ZIP y JPEG2000 sin pérdida. De dicho experimento obtuvimos que para todos los casos los bits por píxel se encuentran por debajo del óptimo teórico (la entropía), puesto que los algoritmos de compresión se ven beneficiados por las redundancias entre los píxeles.

Capítulo 6
Conclusiones

El presente estudio no sólo da cuenta de un análisis y una comparación de los algoritmos de compresión más relevantes en la literatura científica, sino que además propone un algoritmo de compresión que, gracias a que integra los dos tipos de compresión (con y sin pérdida) y se vale de las características de las mastografías, mantiene la sección de interés (ROI, la mama) sin pérdida de información para el diagnóstico. Además, al comparar nuestra propuesta con JPEG2000 hace que RAR lo superé en TC con un 15 % y, también, nuestra propuesta supera a un algoritmo similar con 17 %. Con estas afirmaciones, concluimos que es posible comprimir mastografías digitales a una mayor tasa de compresión que con los métodos de compresión optimizados para imágenes, lo que valida nuestra hipótesis inicial.

Ahora bien, existe la posibilidad de mantener la información relevante para el diagnóstico incluso después de aplicar compresión con pérdida en la sección de interés, lo que, naturalmente, incrementaría la TC. Empero, para determinar si la región de interés pierde o no información relevante para el diagnóstico, tendríamos que consultar la opinión de un experto y, de momento, no contamos con dicho recurso, además de que tal acción, incorporar ambas evaluaciones, se saldría del alcance de una tesis de maestría. Aunque sin duda se trata de un tema que vale la pena investigar y, por lo tanto, lo proponemos como un área de oportunidad para una futura investigación.

Anexo

Resultados de los Algortimos de Compresión

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb001	6.49	5.32	**8.13**	7.08
mdb002	5.40	4.40	**6.57**	5.50
mdb003	5.72	4.60	**7.07**	5.80
mdb004	5.27	4.29	**6.59**	5.45
mdb005	3.97	3.34	**4.80**	4.73
mdb006	3.99	3.34	**4.86**	4.65
mdb007	5.01	4.15	**6.20**	5.74
mdb008	4.23	3.52	**5.23**	4.83
mdb009	5.42	4.51	**6.67**	5.05
mdb010	6.10	5.04	**7.51**	6.82
mdb011	6.07	5.07	**7.42**	5.42
mdb012	5.27	4.40	**6.54**	5.02
mdb013	5.58	4.66	**6.74**	6.44
mdb014	5.11	4.24	**6.35**	5.85
mdb015	6.47	5.40	**7.94**	6.85
mdb016	5.74	4.78	**7.23**	6.72
mdb017	9.05	7.46	**11.05**	8.50
mdb018	8.85	7.32	**10.87**	8.26
mdb019	3.71	3.07	**4.62**	4.70
mdb020	4.11	3.43	**5.04**	4.82
mdb021	4.63	3.80	**5.73**	4.77
mdb022	5.80	4.78	**7.30**	6.08
mdb023	4.77	3.95	**5.86**	4.83
mdb024	5.53	4.60	**6.79**	5.06
mdb025	3.80	3.14	**4.74**	4.49
mdb026	4.20	3.49	**5.21**	4.82
mdb027	4.33	3.64	**5.30**	4.74
mdb028	4.22	3.49	**5.26**	4.73
mdb029	5.79	4.79	**7.01**	5.09
mdb030	5.80	4.80	**7.13**	6.08
mdb031	4.74	3.95	**5.91**	4.71
mdb032	4.14	3.43	**5.27**	4.55

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb033	9.85	8.15	**12.16**	8.40
mdb034	8.56	6.96	**10.60**	8.14
mdb035	9.18	7.47	**11.37**	8.42
mdb036	8.85	7.20	**10.99**	8.25
mdb037	8.52	6.95	**10.46**	8.41
mdb038	8.13	6.71	**10.08**	9.27
mdb039	10.04	8.14	**12.42**	9.98
mdb040	12.71	10.38	**15.89**	13.80
mdb041	5.82	4.75	**7.30**	6.32
mdb042	5.36	4.40	**6.74**	5.94
mdb043	8.86	7.36	**11.04**	8.18
mdb044	7.73	6.40	**9.54**	7.73
mdb045	5.70	4.68	**6.94**	6.09
mdb046	5.47	4.56	**6.82**	5.87
mdb047	6.65	5.48	**8.26**	6.48
mdb048	6.13	5.07	**7.54**	6.30
mdb049	4.15	3.42	**5.14**	5.10
mdb050	4.30	3.57	**5.38**	5.06
mdb051	5.15	4.25	**6.41**	4.95
mdb052	6.65	5.52	**8.37**	6.42
mdb053	10.98	9.02	**12.62**	9.01
mdb054	11.06	9.08	**13.18**	8.87
mdb055	6.62	5.46	**8.27**	7.97
mdb056	7.50	6.20	**9.45**	9.23
mdb057	4.57	3.80	**5.57**	5.42
mdb058	5.05	4.19	**6.17**	5.77
mdb059	7.50	6.42	**9.02**	7.19
mdb060	7.61	6.50	**9.19**	7.30
mdb061	11.77	9.73	**14.32**	8.52
mdb062	8.99	7.52	**11.05**	8.00
mdb063	5.70	4.69	**7.19**	6.66
mdb064	5.79	4.75	**7.28**	6.99
mdb065	9.87	7.99	**12.27**	7.00
mdb066	7.81	6.48	**9.73**	6.52
mdb067	4.44	3.59	**5.50**	4.61
mdb068	5.65	4.56	**6.98**	5.02
mdb069	4.28	3.57	**5.38**	4.58

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb070	5.04	4.22	**6.25**	4.84
mdb071	7.02	5.79	**8.81**	8.65
mdb072	8.19	6.76	**10.27**	9.62
mdb073	7.40	6.05	**9.24**	9.16
mdb074	7.94	6.51	**10.01**	9.91
mdb075	5.95	4.91	**7.04**	6.14
mdb076	6.06	5.08	**7.33**	6.13
mdb077	3.92	3.27	**4.80**	4.74
mdb078	4.91	4.12	**6.11**	5.52
mdb079	5.90	4.88	**7.15**	6.21
mdb080	6.46	5.36	**7.98**	6.30
mdb081	4.11	3.42	**5.16**	4.50
mdb082	4.16	3.50	**5.17**	4.49
mdb083	4.34	3.62	**5.40**	4.94
mdb084	3.79	3.17	**4.69**	4.52
mdb085	5.96	4.91	**7.48**	6.92
mdb086	5.89	4.89	**7.46**	7.09
mdb087	6.04	5.06	**7.39**	6.83
mdb088	5.99	5.01	**7.38**	6.95
mdb089	7.00	5.76	**8.77**	7.80
mdb090	6.04	4.95	**7.61**	7.31
mdb091	6.64	5.51	**8.31**	6.19
mdb092	6.71	5.58	**8.47**	6.25
mdb093	6.15	5.05	**7.76**	7.30
mdb094	5.75	4.81	**7.20**	6.13
mdb095	5.17	4.28	**6.46**	6.16
mdb096	5.71	4.76	**7.03**	6.52
mdb097	4.23	3.56	**5.09**	5.08
mdb098	4.21	3.57	**5.08**	4.98
mdb099	5.60	4.63	**6.91**	6.81
mdb100	5.43	4.48	**6.76**	6.72
mdb101	6.22	5.05	**7.83**	6.41
mdb102	7.54	6.09	**9.55**	6.89
mdb103	5.59	4.60	**7.10**	6.30
mdb104	6.61	5.45	**8.32**	7.18
mdb105	4.55	3.67	**5.69**	5.58
mdb106	4.27	3.50	**5.36**	4.52

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb107	4.12	3.33	**5.29**	5.21
mdb108	4.11	3.34	**5.22**	5.12
mdb109	4.71	3.89	**5.90**	5.37
mdb110	4.93	4.07	**6.19**	5.92
mdb111	4.72	3.91	**5.85**	5.15
mdb112	4.40	3.64	**5.56**	5.08
mdb113	7.91	6.55	**9.74**	7.79
mdb114	10.11	8.35	**12.51**	9.49
mdb115	3.78	3.10	**4.79**	4.37
mdb116	4.72	3.89	**6.01**	4.87
mdb117	5.56	4.59	**6.98**	5.47
mdb118	5.83	4.81	**7.39**	5.99
mdb119	5.14	4.20	**6.42**	4.81
mdb120	5.28	4.33	**6.53**	4.93
mdb121	4.68	3.85	**5.90**	4.72
mdb122	4.77	3.94	**5.94**	4.79
mdb123	6.49	5.37	**7.99**	6.14
mdb124	7.49	6.14	**9.08**	6.58
mdb125	4.57	3.74	**5.78**	5.75
mdb126	4.31	3.54	**5.45**	5.25
mdb127	6.58	5.46	**8.24**	7.77
mdb128	6.96	5.79	**8.86**	8.34
mdb129	4.35	3.60	**5.43**	5.19
mdb130	4.17	3.44	**5.25**	5.15
mdb131	3.49	2.94	**4.25**	3.71
mdb132	3.47	2.92	**4.25**	3.63
mdb133	2.60	2.20	**3.12**	2.87
mdb134	2.57	2.18	**3.12**	2.91
mdb135	3.05	2.51	**3.73**	3.60
mdb136	3.24	2.71	**3.99**	3.84
mdb137	4.75	3.88	**6.07**	5.78
mdb138	4.93	4.02	**6.33**	5.96
mdb139	3.45	2.90	**4.19**	3.70
mdb140	3.19	2.70	**3.83**	3.50
mdb141	2.98	2.49	**3.65**	3.69
mdb142	3.07	2.57	**3.73**	3.76
mdb143	3.34	2.81	**4.00**	3.10

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb144	3.01	2.53	**3.71**	3.06
mdb145	3.48	2.85	**4.39**	4.01
mdb146	3.55	2.89	**4.52**	4.12
mdb147	3.22	2.72	**3.92**	3.07
mdb148	3.26	2.67	**4.15**	3.16
mdb149	3.34	2.83	**4.12**	3.83
mdb150	3.57	3.05	**4.38**	4.11
mdb151	2.25	1.90	**2.68**	2.89
mdb152	2.29	1.90	**2.78**	2.99
mdb153	3.75	3.12	**4.49**	3.28
mdb154	4.15	3.45	**5.06**	3.36
mdb155	3.92	3.30	**4.76**	4.31
mdb156	4.05	3.39	**5.01**	4.45
mdb157	5.28	4.39	**6.57**	6.23
mdb158	5.34	4.44	**6.67**	6.29
mdb159	4.19	3.45	**5.18**	5.15
mdb160	4.21	3.51	**5.18**	5.18
mdb161	5.01	4.12	**6.28**	6.09
mdb162	5.32	4.38	**6.71**	6.78
mdb163	4.58	3.69	**5.71**	5.49
mdb164	4.69	3.79	**5.83**	5.80
mdb165	7.97	6.63	**9.83**	9.35
mdb166	7.74	6.43	**9.67**	9.68
mdb167	4.60	3.83	**5.69**	5.39
mdb168	4.11	3.44	**5.05**	4.86
mdb169	5.64	4.60	**6.88**	7.17
mdb170	5.95	4.85	**7.50**	7.53
mdb171	4.95	3.93	**5.96**	6.34
mdb172	5.30	4.28	**6.61**	6.37
mdb173	3.93	3.26	**4.82**	4.42
mdb174	3.82	3.15	**4.78**	4.43
mdb175	5.71	4.67	**7.13**	6.97
mdb176	6.05	4.96	**7.66**	7.54
mdb177	8.14	6.66	**10.18**	9.33
mdb178	7.20	5.83	**9.12**	8.99
mdb179	12.00	9.50	**13.80**	13.27
mdb180	8.76	7.00	**10.30**	11.24

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb181	5.84	4.83	**7.26**	6.76
mdb182	5.46	4.53	**6.84**	6.79
mdb183	4.03	3.31	**5.02**	4.83
mdb184	4.18	3.44	**5.22**	4.96
mdb185	6.23	5.14	**7.48**	7.09
mdb186	6.98	5.75	**8.66**	7.27
mdb187	5.66	4.69	**7.13**	5.88
mdb188	5.85	4.84	**7.36**	5.98
mdb189	3.93	3.33	**4.84**	4.52
mdb190	4.45	3.80	**5.46**	4.70
mdb191	8.08	6.56	**9.41**	8.01
mdb192	7.01	5.75	**8.64**	7.53
mdb193	4.10	3.35	**5.09**	4.91
mdb194	4.74	3.90	**6.02**	5.11
mdb195	4.18	3.49	**5.11**	4.63
mdb196	3.82	3.20	**4.71**	4.43
mdb197	5.57	4.57	**6.69**	6.26
mdb198	5.57	4.61	**6.85**	6.17
mdb199	5.54	4.59	**6.86**	6.19
mdb200	5.39	4.43	**6.86**	6.20
mdb201	3.64	2.99	**4.45**	4.52
mdb202	3.98	3.30	**4.99**	4.62
mdb203	4.23	3.54	**5.18**	4.71
mdb204	4.26	3.56	**5.30**	4.78
mdb205	4.14	3.46	**5.18**	4.91
mdb206	4.41	3.69	**5.48**	4.99
mdb207	6.98	5.70	**8.69**	6.98
mdb208	6.37	5.21	**8.02**	6.97
mdb209	5.27	4.37	**6.61**	5.12
mdb210	4.94	4.16	**6.14**	5.02
mdb211	5.26	4.32	**6.59**	6.06
mdb212	5.65	4.65	**7.12**	6.28
mdb213	7.12	5.88	**8.91**	7.84
mdb214	7.37	6.10	**9.28**	7.98
mdb215	4.03	3.27	**5.06**	4.95
mdb216	4.59	3.69	**5.76**	5.40
mdb217	5.11	4.18	**6.27**	5.06

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb218	5.19	4.26	**6.27**	5.06
mdb219	3.82	3.16	**4.79**	4.62
mdb220	3.93	3.26	**4.91**	4.72
mdb221	6.16	5.07	**7.68**	6.47
mdb222	5.83	4.81	**7.36**	6.35
mdb223	7.59	6.31	**9.40**	7.72
mdb224	7.52	6.27	**9.51**	7.84
mdb225	6.83	5.64	**8.56**	6.74
mdb226	6.85	5.68	**8.69**	6.84
mdb227	5.71	4.70	**7.14**	7.12
mdb228	5.97	4.92	**7.51**	7.31
mdb229	5.01	4.15	**6.25**	5.69
mdb230	5.04	4.16	**6.30**	5.65
mdb231	3.74	3.20	**4.58**	4.39
mdb232	4.14	3.52	**5.13**	4.60
mdb233	5.77	4.73	**7.27**	6.57
mdb234	8.19	6.67	**10.32**	6.84
mdb235	4.71	3.87	**5.89**	4.79
mdb236	5.14	4.26	**6.41**	5.06
mdb237	5.73	4.75	**6.86**	5.98
mdb238	5.67	4.72	**6.93**	6.09
mdb239	4.57	3.70	**5.65**	5.05
mdb240	4.58	3.70	**5.69**	5.21
mdb241	6.81	5.64	**8.23**	7.55
mdb242	6.94	5.78	**8.44**	7.63
mdb243	5.77	4.69	**7.16**	6.27
mdb244	5.12	4.18	**6.47**	6.06
mdb245	6.69	5.49	**8.29**	7.59
mdb246	7.08	5.84	**8.91**	7.87
mdb247	5.20	4.37	**6.36**	5.02
mdb248	5.25	4.41	**6.43**	5.11
mdb249	5.17	4.26	**6.48**	5.96
mdb250	5.76	4.75	**7.20**	6.25
mdb251	5.28	4.42	**6.59**	5.87
mdb252	5.52	4.61	**6.80**	5.95
mdb253	4.52	3.63	**5.54**	4.95
mdb254	4.45	3.63	**5.45**	4.90

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb255	4.64	3.88	**5.63**	4.96
mdb256	4.62	3.87	**5.59**	4.91
mdb257	5.07	4.14	**6.28**	6.25
mdb258	4.91	3.96	**6.08**	5.99
mdb259	4.70	3.82	**5.76**	5.12
mdb260	4.24	3.41	**5.04**	4.98
mdb261	9.32	7.56	**11.40**	8.77
mdb262	8.90	7.31	**10.87**	8.65
mdb263	5.02	4.15	**6.22**	5.83
mdb264	4.81	4.03	**5.97**	5.77
mdb265	5.62	4.57	**6.95**	7.11
mdb266	6.45	5.39	**8.04**	7.44
mdb267	4.15	3.42	**5.16**	4.78
mdb268	5.18	4.25	**6.49**	5.20
mdb269	6.16	5.18	**7.52**	6.13
mdb270	6.73	5.68	**8.25**	6.35
mdb271	4.03	3.37	**4.86**	4.63
mdb272	4.35	3.72	**5.22**	4.70
mdb273	3.98	3.38	**4.83**	4.53
mdb274	3.84	3.21	**4.74**	3.40
mdb275	4.64	3.83	**5.76**	5.81
mdb276	4.52	3.75	**5.65**	5.56
mdb277	5.13	4.25	**6.39**	6.01
mdb278	5.78	4.85	**7.25**	6.28
mdb279	4.02	3.34	**4.99**	4.80
mdb280	2.58	2.05	**2.78**	3.48
mdb281	7.12	5.91	**8.76**	6.86
mdb282	7.29	5.99	**9.17**	7.00
mdb283	5.49	4.48	**6.65**	5.96
mdb284	6.75	5.62	**8.42**	6.78
mdb285	5.15	4.30	**6.41**	5.80
mdb286	5.54	4.59	**6.92**	6.10
mdb287	11.31	9.10	**13.69**	9.05
mdb288	12.68	10.03	**15.28**	9.65
mdb289	7.62	6.20	**9.24**	7.79
mdb290	9.12	7.40	**11.26**	8.58
mdb291	5.03	4.12	**6.30**	5.37

Cuadro 1: Resultados en TC de ACH con 7ZIP, ACH con ZIP, ACH con ZIP y JPEG2000 sin Pérdida.

Imagen	**ACH - 7ZIP**	**ACH - ZIP**	**ACH - RAR**	**JPEG2000**
mdb292	5.11	4.19	**6.49**	5.37
mdb293	3.90	3.24	**4.82**	4.75
mdb294	4.28	3.62	**5.25**	4.83
mdb295	4.42	3.60	**5.52**	5.21
mdb296	4.20	3.47	**5.35**	5.19
mdb297	4.52	3.77	**5.53**	5.62
mdb298	4.69	3.93	**5.83**	5.67
mdb299	4.06	3.42	**4.97**	4.61
mdb300	4.40	3.72	**5.37**	4.89
mdb301	5.67	4.80	**6.98**	6.23
mdb302	6.83	5.75	**8.55**	7.49
mdb303	5.93	5.02	**7.21**	6.13
mdb304	6.12	5.21	**7.47**	6.22
mdb305	5.11	4.23	**6.28**	5.73
mdb306	5.65	4.70	**6.98**	6.11
mdb307	4.24	3.54	**5.19**	4.60
mdb308	4.57	3.80	**5.66**	4.85
mdb309	3.94	3.31	**4.73**	4.65
mdb310	4.17	3.54	**4.95**	4.62
mdb311	4.39	3.59	**5.37**	5.07
mdb312	5.33	4.40	**6.59**	5.36
mdb313	4.44	3.76	**5.35**	4.64
mdb314	4.28	3.62	**5.14**	4.64
mdb315	4.78	3.90	**6.09**	5.08
mdb316	4.53	3.73	**5.77**	4.97
mdb317	9.11	7.50	**10.48**	8.37
mdb318	8.88	7.31	**10.33**	8.39
mdb319	3.95	3.27	**4.89**	4.56
mdb320	4.05	3.32	**5.00**	4.74
mdb321	5.54	4.55	**6.77**	6.01
mdb322	5.95	4.88	**7.36**	6.47
Promedio	5.55	4.58	**6.86**	5.98

Bibliografía

[1] T. Ebrahimi A. Skodras, C. Christopoulos. The JPEG 2000 Still Image Compression Standard. *Signal Processing Magazine, IEEE*, 18(5):36–58, 2001.

[2] T. Acharya and P.S. Tsai. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Wiley, 2005.

[3] M. Adler, T. Boutell, J. Bowler, C. Brunschen, A. M. Costello, L. D. Crocker, A. Dilger, Oliver O. Fromme, J. Gailly, C. Herborth, A. Jakulin, N. Kettler, T. Lane, A. Lehmann, C. Lilley, D. Martindale, O. Mortensen, K. S. Pickens, R. P. Poole, G. Randers-Pehrson, G. Roelofs, W. V. Schaik, G. Schalnat, P. Schmidt, M. Stokes, T. Wegner, and J. Wohl. Portable Network Graphics. Technical report, Oxford Brookes University, November 1996.

[4] Adobe Developers Association. TIFF 6.0 Specifications. Technical report, Adobe Systems Incorporated, June 1992.

[5] A. Akansu and R. Haddad. *Multiresolution Signal Decomposition: Transforms, Subbands, and Wavelets*. Academic Press, 2001.

[6] S. Astley, D. Betal, N. Cerneaz, D.R. Dance, S.L Kok, J. Parker, I. Ricketts, J. Savage, E. Stamatakis, and P. Taylor. The Mammographic Image Analysis Society Digital Mammogram Database. In *Exerpta Medica. International Congress Series*, volume 1069, pages 375–378, 1994.

[7] C. DeSantis, R. Siegel, P. Bandi, A. Jemal. Breast Cancer Statistics. *CA: Cancer Journal for Clinicians*, 61(6):408–418, 2011.

[8] D.P. Chakraborty. Recent Advances in Observer Performance Methodology: Jackknife Free-Response ROC (JAFROC). *Radiation Protection Dosimetry*, 114(1-3):26–31, 2005.

[9] D. Charlap. The BMP File Format, Part 1. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 20(3):44–51, 1995.

[10] CompuServe Incorporated. Graphics Interchange Format. `http://www.w3.org/Graphics/GIF/spec-gif87.txt`. Último acceso 03-11-2014.

[11] R. Eugene. WinRAR. `http://www.winrar.es`. Último acceso 03-11-2014.

[12] W. F. Good, J. H. Sumkin, M. Ganott, L. Hardesty, B. Holbert, C. M. Johns, and A. H. Klym. Detection of Masses and Clustered Microcalcifications on Data Compressed Mammograms: An Observer Performance Study. *American Journal of Roentgenology*, 175(6):1573–1576, 2000.

[13] E. Hamilton. JPEG File Interchange Format. Technical report, C-Cube Microsystems, Septiembre 1992.

[14] D.A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE (The Institute of Radio Engineers)*, 40(9):1098–1101, 1952.

[15] CompuServe Incorporated. Graphics Interchange Format. `http://www.w3.org/Graphics/GIF/spec-gif89a.txt`. Último acceso 03-11-2014.

[16] S. Jayaraman, S. Esakkirajan, and T. Veerakumar. *Digital Image Processing*. Tata McGraw-Hill Education, First edition, 2011.

[17] N. Karimi, S. Samavi, E. Mahmoodzadeh, and S. Shirani. Lossless Compression of Mammographic Images with Region-based Predictor Selection. *Transactions on Electrical and Electronic Engineering*, 8(5):478–482, 2013.

[18] A. Khademi and S. Krishnan. Comparison of JPEG2000 and other Lossless Compression Schemes for Digital Mammograms. In *Proceedings of 27th Annual*

International Conference of the Engineering in Medicine and Biology Society, pages 3771–3774. IEEE, 2006.

[19] M. S. R. Kumar, S. Koliwad, and G. S. Dwarakish. Lossless Compression of Digital Mammography Using Fixed Block Segmentation and Pixel Grouping. In *Proceedings of the 2008 Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, pages 201–206. IEEE Computer Society, 2008.

[20] A. E. Laemmel. Coding Processes For Band-width Reduction in Picture Transmission. In *Proceedings of the Institute of Radio Engineers*, volume 39, pages 293–293, 1951.

[21] P. Lindner. Registration of a new MIME Content Type-/Subtype. Technical report, Team Gopher, 1993.

[22] J. Miano. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. Addison-Wesley Professional, 1999.

[23] R. K. Mulemajalu and S. Koliwad. Lossless Compression of Digital Mammography using base Switching Method. *Journal of Biomedical Science and Engineering*, 2:336–344, 2009.

[24] I. Pavlov. 7ZIP. *URL http://www.7-zip.org*, 2013.

[25] M. Penedo, M. Lado, P. G. Tahoces, M. Souto, and J. J. Vidal. Effects of JPEG2000 Data Compression on an Automated System for Detecting Clustered Microcalcifications in Digital Mammograms. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):354–361, 2006.

[26] A. Said and W. A. Pearlman. A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, 1996.

[27] K. Sayood. *Introduction to Data Compression*. Newnes, 2012.

[28] C. E. Shannon. A Mathematical Theory of Communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[29] J. M. Shapiro. Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[30] Y.Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards, Second Edition*. Image Processing Series. Taylor & Francis, 2008.

[31] American Cancer Society. Breast Cancer. `http://www.cancer.gov/espanol/pdq/deteccion/seno/Patient/page3`. Último acceso 03-11-2014.

[32] D. Speck. Proposal for Next Generation Lossless Compression of Continuous-Tone Still Pictures: Activity Level Classification Model (ALCM). *ISO Working Document*, 1995.

[33] J. A. Storer and T. G. Szymanski. Data Compression Via Textual Substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[34] M. Tayel and A. Mohsen. Breast Boarder Boundaries Extraction using Statistical Properties of Mammogram. In *Proceeding of the International Conference on Signal Processing*, pages 2468–2471. IEEE, 2010.

[35] M. Tayel and A. Mohsen. Hybrid Lossless Compression of Breast Mammography. In *Proceedings of the Workshop on Signal Processing Systems*, pages 273–277. IEEE, 2011.

[36] G. K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4):30–44, 1991.

[37] M. J. Weinberger, G. Seroussi, and G. Sapiro. The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS. *IEEE Transactions on Image Processing*, 9(8):1309–1324, 2000.

[38] T. A. Welch. High Speed Data Compression and Decompression Apparatus and Method, December 10 1985. US Patent 4,558,302.

[39] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, 30(6):520–540, 1987.

[40] X. Wu and N. Memon. Context-Based, Adaptive, Lossless Image Coding. *IEEE Transactions on Communications*, 45(4):437–444, 1997.

[41] P. Xu, Zuo, Y. Xu W. Xu, and H. Chen. A New Diagnosis Loseless Compression Method for Digital Mammography Based on Multiple Arbitrary Shape ROIs Coding Framework. *International Journal of Modern Education and Computer Science*, 3(5):33, 2011.

[42] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[43] J. Ziv and A. Lempel. Compression of Individual Sequences Via Variable-rate Coding. *Information Theory, IEEE Transactions on*, 24(5):530–536, 1978.