CAPÍTULO 1

INTRODUCCIÓN

Cuando se muestrea y cuantifica una función bidimensional de intensidad para crear una imagen digital se produce una enorme cantidad de datos. Usar esta gran cantidad de datos para realizar un procesamiento de la señal posterior sería impensable. La primera necesidad con la que se encuentra pues, el procesamiento de imágenes, es reducir el volumen de datos que representen a una misma información. El problema de reducción de los datos está basado en la eliminación de los datos redundantes de la imagen. Matemáticamente hablando, esto equivale a transformar el conjunto de datos en otro conjunto de datos sin correlacionar.

Los primeros estudios en el campo de la compresión tienen su origen en el mundo analógico. Estos estudios iban dirigidos a la reducción del ancho de banda de transmisión. Con la evolución de la tecnología y el paso de testigo del mundo analógico al mundo digital, junto con la aparición de circuitos digitales avanzados, se traslada los estudios de compresión al procesamiento digital de imágenes.

La evolución de los estudios de compresión ha ido siempre ligada al crecimiento de la informática multimedia. Aparte de esta unión, las aplicaciones en las que la compresión de imágenes desempeña un papel fundamental son tan numerosas como diversas. La encontramos en videoconferencias, sensores remotos, imágenes de documentos, imágenes médicas, la transmisión de facsímiles (fax), el control remoto de vehículos en numerosas aplicaciones militares, espaciales y de manipulación de materias peligrosas. Estas aplicaciones dependen de la eficaz manipulación, almacenamiento y transmisión de imágenes en escala de grises o de color.

En los comienzos existían abundantes estudios acerca de la compresión de imágenes en blanco y negro. Con la creciente aparición de las tecnologías de color se hizo necesaria esa migración. Dicho cambio iba a suponer no un simple hecho de comprimir tres planos de color en vez de uno, como especificaba el estándar JPEG, sino

un cambio de total y diferenciador a la hora de tratar las imágenes a color. En las imágenes a color se investigarán nuevas fuentes de redundancia, como es la redundancia intrapíxel, que llegarán a la conclusión que comprimir una imagen de color es más complicado que comprimir tres planos independientes, si queremos obtener buenos resultados. El estándar JPEG-LS[1] es un compresor de imágenes de color, cuya eficacia supera al anterior JPEG. Otro esquema de compresión de imágenes de color es el CALIC[2].

Existen ocasiones en las que tenemos un tope para la representación de una cierta cantidad de información. Estamos hablando de que la información que recibamos comprimida no será exactamente la misma que la original, porque para no superar dicho tope hemos reducido la información. Este es el caso de compresión de imágenes con pérdidas de información. Existen casos en los que dicho tipo de compresión es impensable, como ejemplo tenemos las imágenes adquiridas por satélite donde el coste de la información es muy alto como para que se pierda en el proceso de compresión. La razón de las imágenes por satélite está clara, pero otro tipo de imágenes como las imágenes médicas, no se puede permitir unas pérdidas ya que podría llevar a una confusión en un diagnóstico. No sólo son aplicables al diagnóstico, sino también al telediagnóstico donde la compresión sin pérdidas sería aún más importante. Otro tipo de imágenes que necesitan de su compresión sin pérdidas son las imágenes de alta resolución y los documentos importantes que se quieran digitalizar.

Viendo la importancia que tiene en algunos campos la compresión de imágenes sin pérdidas, se deben afrontar estudios en aras de mejorar las tasas de compresión de los actuales algoritmos e intentar reducir en lo posible el coste computacional de éstos.

En este proyecto se van a desarrollar y analizar la ganancia en compresión de distintos esquemas de compresión sin pérdidas. El objetivo último es encontrar un algoritmo de compresión que mejore al JPEG-LS con el mínimo coste computacional posible. Se van a desarrollar y analizar la ganancia en compresión de distintos esquemas de compresión sin pérdidas sobre imágenes a color con 24bits/píxel, 8bits para cada plano de color. Los esquemas que se irán analizando dependerán de cómo se vayan comportando los esquemas previos, es decir, cuando modifiquemos un bloque del esquema de compresión lo compararemos con el anterior esquema y decidiremos cuál de ellos escoger para el siguiente esquema en función de los resultados que obtengamos. Los bloques que iremos probando tendrán mucho que ver con las actuales tendencias de los algoritmos de compresión sin pérdidas, como el JPEG-LS y el algoritmo SLCIC [Serrano,01]. Entre algunos podemos citar la segmentación de la imagen previa a la compresión, el uso de predictores, el uso de codificación aritmética, de la cual estudiaremos su viabilidad con contexto y sin él, el uso de predicción de contextos basado en gradientes locales…. La inclusión de cada uno de ellos llevará asociado un estudio de tasas de compresión con el fin de poder decidir si proseguir por ese camino incluyendo ese bloque o por el contrario optar por el anterior que usábamos y seguir probando otras opciones.

La estructura de este proyecto posee 8 capítulos. En el capítulo 2 se describen los fundamentos más importantes de la imagen digital, centrándose en las imágenes de color. A continuación en el 3 se hace una descripción de la teoría de la información

[1] JPEG-LS : Joint Picture Expert Group- Lossless
[2] Context-based Adaptative Lossless Image Codec

aplicada a las imágenes, para continuar en el 4 con los fundamentos sobre compresión, básicos para el desarrollar cualquier algoritmo. El capítulo 5 está dedicado a la descripción de dos estándares de compresión el JPEG-LS y el JBIG, y al algoritmo de compresión de imágenes en blanco y negro basado en segmentación, el SLIC[3]. En el capítulo 6 se describen los algoritmos desarrollados y se ordenan partiendo del algoritmo inicial, al cual se le van añadiendo en cada punto nuevos bloques y quitando los que no dan buenos resultados, hasta llegar al último. El capítulo 7 es dónde se comparan los resultados obtenidos en cada uno de los esquemas de compresión que hemos desarrollado, a su vez, se comparan también con algoritmos como SLCIC[4], JPEG-LS y JBIG. Por último expondremos en el capítulo 8 las conclusiones extraídas de este proyecto y las líneas futuras de investigación que se proponen.

[3] Segmentation-based Lossles Image Compression
[4] Segmentation-based Lossles Color Image Compresión

CAPÍTULO 2

FUNDAMENTOS DE LA IMAGEN DIGITAL

El término imagen se refiere a una función bidimensional de intensidad de luz *f(x,y)* donde *x* e *y* son las coordenadas espaciales y *f* es la función de nivel de gris o de una de las componentes de color en ese punto, dependiendo si la imagen es en escala de grises o a color. Considerando que la imagen en este caso está en escala de grises dicha función *f(x,y)* representará en los niveles más altos a las zonas más brillantes, y por el contrario los niveles más bajos a las zonas más oscuras.

El esquema general de un procesado de imágenes es el que se muestra en la Figura 2-1.

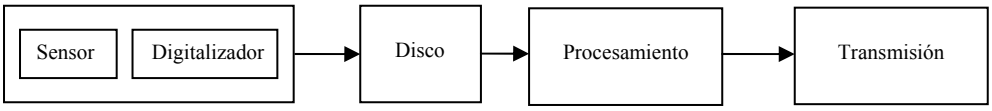


Figura 2-1: Esquema general de un procesado digital de imágenes

Como se puede observar, el primer dispositivo que empleamos para realizar el procesado de la imagen es el sensor, a éste le llega la intensidad de luz denominada anteriormente como *f(x,y)*, esta intensidad de luz debe cumplir que

$$0 < f(x,y) < \infty$$

puesto que estamos hablando de una energía, por lo tanto, tiene que ser finita. Dicha intensidad está compuesta a su vez de dos componentes

$$f(x,y) = i(x,y) \cdot r(x,y) \rightarrow \begin{cases} i(x,y) \rightarrow ILUMINACIÓN \\ r(x,y) \rightarrow REFLECTANCIA \end{cases}$$

la iluminación depende de la fuente de luz que esté iluminando el objeto, y la reflectancia está determinada por la naturaleza de los objetos que están siendo iluminados por la fuente de luz. Al igual que la intensidad estas dos funciones también están acotadas

$$0 < r(x,y) < 1$$
$$0 < i(x,y) < \infty$$

la reflectancia puede variar entre 0, que quiere decir que el objeto no reflecta nada, y 1, cuando toda la luz es reflectada. La iluminación al estar en íntima relación con la fuente de luz, se debe comportar como forma de energía, es por ello que debe ser finita, al igual que lo es la intensidad ya que es producto de estas dos funciones. [Castleman, 96]

2.1 REPRESENTACIÓN DE LA IMAGEN DIGITAL

Si nos centramos en la función de intensidad *f(x,y),* nos damos cuenta que dicha función bidimensional tiene infinitos intervalos espaciales tanto en *x* como en *y,* así como el valor de la misma función, por ello encontraría problemas a la hora de ser almacenada para realizarle un procesado posterior. Para pasar dicha función al campo digital tendríamos que hacer dos procesos, uno de ellos llamado *muestreo espacial* en el cual discretizamos los ejes de la imagen, el *x* y el *y*, y el valor que usaremos para la imagen, será aquél que corresponda a una *x* e *y* válidas según el proceso llevado anteriormente a cabo. Con esto conseguimos la resolución de la imagen, dependiendo de cómo hayamos discretizado las componentes espaciales, la resolución será distinta. Por ejemplo, si a una imagen cuadrada le discretizamos las componentes espaciales con 256 puntos cada componente, decimos que la imagen tiene una resolución de 256x256. A parte de las componentes espaciales, tenemos también la propia función *f(x,y)*, que también es continua, por ello también ha de ser discretizada. Aquí hay que hacer una distinción entre imágenes a color o imágenes en B/N, en las primeras debemos discretizar cada uno de los planos de color, y en las segundas sólo discretizamos el único plano de la que se compone. A este proceso se le denomina *cuantificación,* y así como el muestreo espacial nos daba como resultado una determinada resolución, aquí lo que obtenemos es un determinado número de niveles de cada componente, o de una si estamos hablando de imágenes en B/N. El número de bits/píxel se calcula así

$$n^{o}\, bits / pixel = \log_2 número\ niveles$$

el tamaño de total de la imagen es

$$TAMAÑO = M \times N \times B$$

donde el producto de MxN es la resolución de la imagen y B es el número de bits/píxel.

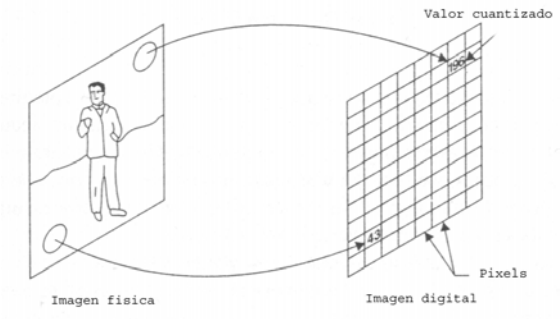


Figura 2-2 : Ejemplo de muestreo espacial y cuantificación

Estudios realizados han llegado a la conclusión que los humanos somos poco sensibles a los niveles de gris, por lo tanto si debemos elegir entre que una imagen tenga poca resolución y altos numero de bits/píxel, o entre que tenga alta resolución y bajo número de bits/píxel, debemos elegir la segunda opción si estamos hablando de un mismo tamaño para los dos.[González, 96]

Se han ideado también formas de muestreo y cuantificación un poco más elaboradas teniendo en cuenta como actúa el cerebro humano ante la imagen, como aplicar distinta resolución a diferentes zonas de la imagen, por ejemplo.

2.2 FUNDAMENTOS DEL COLOR

La luz es una radiación electromagnética que al excitar nuestro sistema visual crea una sensación llamada color. Nos estamos refiriendo a la parte de la luz que es visible, esto es, aquella que está comprendida entre los 400nm y los 700nm de longitud de onda, dejando fuera las frecuencias ultravioletas y las infrarrojas que no excitan y, que por otra parte en cantidades excesivas, pueden dañar el ojo; de filtrar parte de estas longitudes de onda dañinas se encargan las proteínas que forman la estructura del cristalino. En la Figura 2-3 vemos la estructura casi esférica que posee el ojo humano de un diámetro medio aproximado de 20mm.

El ojo está rodeado por tres membranas denominadas la *córnea* y la *esclerótica*, que constituyen la cubierta exterior, la *coroides* y la *retina*. La retina es la membrana más interna del ojo. Cuando un objeto esta enfocado correctamente, la luz de un objeto exterior al ojo forma su imagen en la retina. La visión después de que la imagen del objeto exterior está formada en la retina se debe a una distribución de dos clases de receptores, cada uno de los cuales tiene una función. Las dos clases de receptores son: los *conos* y los *bastones*.

Los conos suman un número aproximado de entre 6 y 7 millones, la mayoría de los cuales está concentrado en la región central de la retina llamada *fóvea*; éstos se encargan de la percepción del color. Son unos receptores muy sensibles al color lo que otorga a la visión humana la posibilidad de apreciar detalles muy finos. Los conos poseen cada uno de ellos una terminación nerviosa. La visión mediante los conos se denomina *fotópica* o visión de luz brillante.

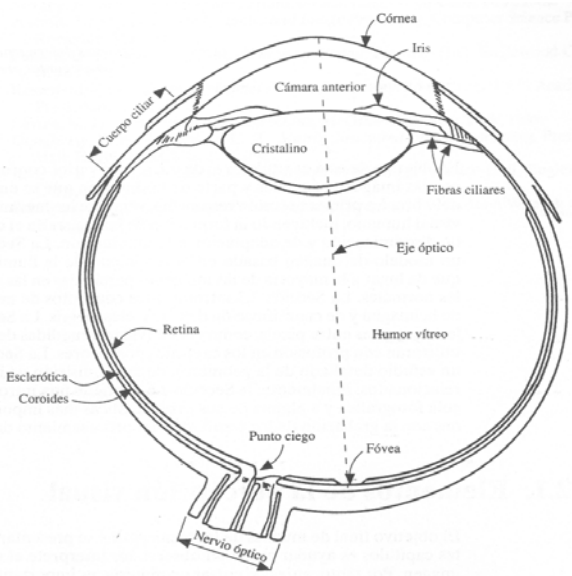


Figura 2-3 : Ojo Humano

Los bastones son el otro tipo de receptores cuyo número puede variar entre 75 y 150 millones, un número muy superior al de los conos, y están repartidos por toda la superficie retiniana. Éstos, debido a que un conjunto de ellos comparten una misma terminación nerviosa y a su mayor distribución, reducen la cantidad de detalle. Los bastones sirven para dar una visión general del campo de visión, y no en la percepción del color. La visión de estos receptores de denomina *escotópica* o visión tenue.

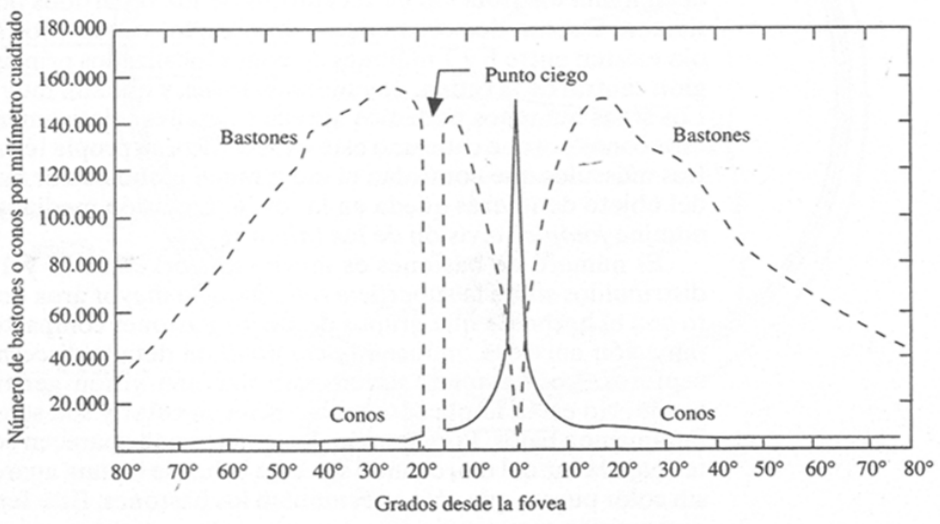


Figura 2-4 : Distribución de bastones y conos en la retina humana.

Los conos, responsables de la percepción del color, podemos dividirlos en tres tipos. Después de haber realizado numerosos experimentos acerca del color, se ha demostrado que los humanos perciben el color según la *teoría tricromática* postulada

por Thomas Young en 1802, que explica que la distribución espectral de la energía de la luz se puede descomponer en tres fuentes independientes, para las cuales existe un tipo de cono cuya respuesta a dicha fuente encuentra un máximo en la absorción para esa frecuencia. Las tres absorciones de los distintos conos las podemos observar en la Figura 2-5.[González, 96]

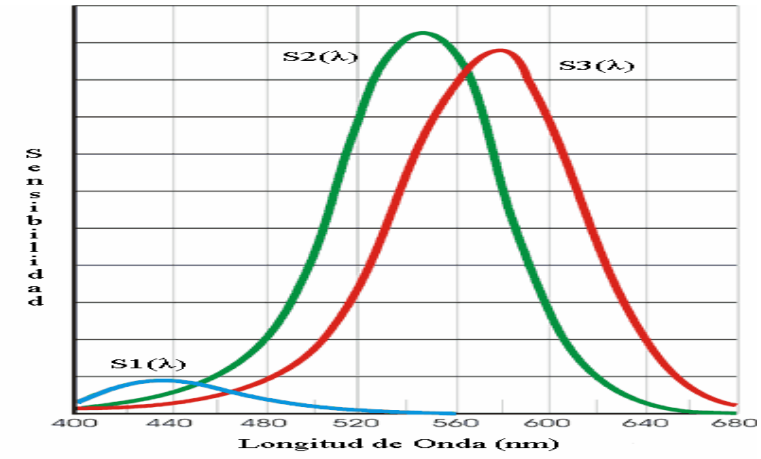


Figura 2-5 : Espectro de absorción de los tres tipos de conos en la retina humana

Cada uno de estos espectros de absorción están centrados en una determinada longitud de onda, que será a la cual dichos conos serán más sensibles. Las longitudes de onda a las que nos referimos son las siguientes:

- $S_1(\lambda) \rightarrow \lambda_1 = 437nm$ (azul).
- $S_2(\lambda) \rightarrow \lambda_2 = 533nm$ (verde).
- $S_3(\lambda) \rightarrow \lambda_3 = 564nm$ (rojo).

Si llamamos $C(\lambda)$ a la densidad espectral de una luz coloreada, ésta producirá una sensación de color que es la combinación de las tres respuestas espectrales de los conos de la retina, cuya respuesta es:

$$\alpha_i(C) = \int_{\lambda_{\min}}^{\lambda_{\max}} S_i(\lambda) C(\lambda)\, d\lambda, \quad i = 1,2,3$$

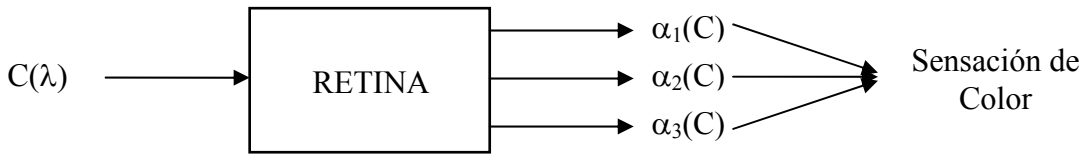


Figura 2-6 : Representación esquemática de la función de la retina.

Al ser la sensación que recibe el cerebro una combinación de las generadas por los tres tipos de conos, cabría la posibilidad de que si tenemos una fuente de luz coloreada con una distribución de energía $C_1(\lambda)$, podríamos encontrar otra fuente de color $C_2(\lambda)$ que cree la misma sensación de color en el cerebro. Ésta es la explicación a que si percibimos una fuente de un color, percibimos lo mismo si percibimos dos colores que mezclados dan el primero; en realidad estamos percibiendo dos fuentes de color que se combinan de forma que la sensación que dan es la misma que la primera.

2.2.1 Sistemas de Representación del Color

Como se ha explicado anteriormente, la estructura del ojo humano hace que los colores se vean como combinaciones de los denominados *tres colores primarios*, que son el rojo (R,Red), verde (G,Green) y azul (B,Blue). Para conseguir una normalización la CIE (*Comisión Internacional de la Iluminación*) designó en 1931 para cada uno de los tres colores primarios las siguientes longitudes de onda:

- $R_{CIE} \rightarrow 700nm.$
- $G_{CIE} \rightarrow 546.1nm.$
- $B_{CIE} \rightarrow 435.8nm.$

Que estos tres colores tengan dichas longitudes de onda no quiere decir que para conseguir cualquier color haya que hacer uso de dichas longitudes de onda, ya que para que se consiga un color cualquiera, la longitud de onda de los colores primarios debe variar. Hay que hacer distinción también entre *colores primarios de luz* y *colores primarios de pigmentos*. Un color primario de pigmento se define como aquel que

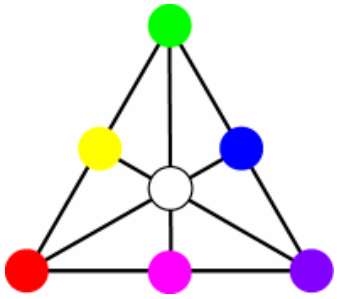


Figura 2-7 : Representación de los colores primarios de luz y de pigmento

sustrae un color primario de luz y refleja los otros dos. La suma de dos colores primarios, ya sean de luz o de pigmento, dan un color secundario de luz o de pigmento. En la figura se observa los colores primarios de luz que son el conjunto llamado RGB (Rojo, Verde y Azul) que se encuentran en los vértices del triángulo, mientras que los colores primarios de pigmento se encuentran en la mitad de los lados del triángulo, que son el Magenta, Cian y Amarillo. Los colores primarios de luz se corresponden con los secundarios de pigmento y viceversa. [González, 96]

Un ejemplo claro de la naturaleza aditiva de los colores de luz lo tenemos en algo tan familiar como es la televisión. En el interior de una televisión a color tenemos una distribución de material fosforescente sensible a los electrones dispuestos según un

esquema triangular. Cada uno de estos puntos que forman un triángulo son sensibles a un cañón de electrones que estimulan que dicho punto, sea rojo, verde o azul emita luz del propio color y que al llegar a nuestra retina por el funcionamiento ya indicado de los conos, percibamos la sensación en conjunto del color.

Las características generalmente empleadas para distinguir un color de otro son el *brillo*, *tono* y *saturación*. El brillo está relacionado con la noción cromática de intensidad. El tono se refiere a la longitud de onda predominante en un color que hace que digamos que algo es rojo, verde, naranja…. La saturación se refiere a la pureza relativa o cantidad de luz blanca mezclada con un tono. Al par tono-saturación se le denomina *cromaticidad*, por lo tanto un color puede caracterizarse por el brillo y la cromaticidad.

En 1964 la CIE estableció otro estándar muy parecido donde las longitudes de onda en las cuales se encontraban los picos serían las mismas, lo que variaría serían la curvas. El problema del sistema RGB_{CIE} es que las curvas tomaban valores negativos, lo cual no era realizable físicamente, por lo tanto se opto por una representación artificial en la que los valores de las curvas eran todos positivos. El sistema se llamó *XYZ* y la relación de transformación entre los sistemas es la que sigue:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.490 & 0.310 & 0.200 \\ 0.177 & 0.810 & 0.011 \\ 0.000 & 0.010 & 0.990 \end{bmatrix} \begin{bmatrix} R_{CIE} \\ G_{CIE} \\ B_{CIE} \end{bmatrix}$$

Los modelos de color empleados en la actualidad son diversos, y la elección de un modelo u otro depende la orientación que se le quiera dar a la aplicación. Los modelos orientados al hardware son el *RGB* para monitores y cámaras de vídeo, y el modelo *CMY* para impresoras en color. Existen otros dedicados para las transmisiones de televisión como es el modelo *YIQ*, en el que se separan las componentes de luminancia de las cromáticas, éstas separadas a su vez en *fase* y *cuadratura*. También existe otro modelo que normalmente se usa para manipulación de imágenes a color, el sistema *HSI*.

2.2.1.1 Modelo de color RGB

En este modelo las componentes del color vienen dadas por las componentes espectrales primarias de rojo, verde y azul como indica las siglas en su traducción inglesa. Este modelo está basado en un sistema de coordenadas cartesianas que podemos representar por el cubo visto en la Figura 2-8.

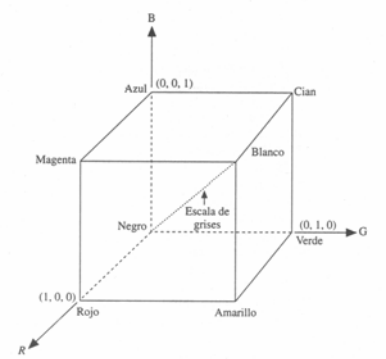


Figura 2-8 : Cubo de valores RGB

Como se puede observar en la Figura 2-8, tres de los vértices tenemos los colores secundarios de luz que son el cian, magenta y amarillo. La línea que une el origen de coordenadas del cubo con el punto (1,1,1) es la línea de escala de grises. Se considera en esta representación que los valores de los colores primarios están normalizados a el intervalo [0,1]. Las imágenes del modelo de color RGB consisten en tres planos de imagen independientes, uno por cada color primario. [González, 96]

Este modelo de color no tiene la información de brillo y la de cromaticidad separadas, por lo tanto al aplicar un algoritmo de tratamiento digital de imágenes habrá que aplicarlo a los tres planos de color independientemente. Si tomamos como ejemplo, la ecualización de un histograma, habrá que ecualizar los tres planos y la información de cromaticidad de la imagen se verá afectada. Por ello, para el tratamiento digital de imágenes se suelen usar otros modelos de color.

2.2.1.2 Modelo de color CMY

Este modelo corresponde a los colores secundarios de luz o los primarios de pigmento que son el cian, magenta y amarillo. Nos encontramos ante un modelo de color cuyo uso básico lo encontramos en la última parte del procesamiento de imagen, esto es, cuando imprimimos la imagen por ejemplo. Estos tres colores secundarios de luz son los que usan las impresoras para imprimir imágenes a color. La transformación de RGB a CMY es la siguiente.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Como vemos, la transformación es muy sencilla y el modelo de color CMY sigue estando normalizado al igual que el RGB. La transformación inversa no tiene interés alguno ya que este cambio de modelo se realiza al final del procesamiento, y por consiguiente no interesa después de imprimir el objeto procesado volverlo a cambiar a su modelo anterior.

2.2.1.3 Modelo de color YIQ

Este modelo de color se usa en las emisiones comerciales de televisión, ya que es muy eficaz a la hora de transmitir imágenes, además de hacer compatible la transmisión de imágenes a color con el funcionamiento de los televisores en B/N, ya que este modelo está formado por *Y*: luminancia, *I*: componente en fase, *Q*: componente cuadratura, estas dos ultimas representan al color y están desacopladas de la anterior. De esta forma los televisores en B/N podían sólo usar la información de la imagen proveniente en la componente *Y* y así poder coexistir ambas tecnologías. La transformación con el modelo RGB es la siguiente.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Este modelo fue diseñado para aprovechar la mayor sensibilidad del sistema visual humano a los cambios de saturación, por lo que se puede transmitir aplicando más ancho de banda o más bits, dependiendo si hablamos de transmisión analógica o digital, a la componente *Y* que a las otras dos. Otra ventaja que este modelo presenta con respecto a el RGB es que se pueden tratar la información de luminancia y la de cromaticidad independientemente sin que un procesado de imagen sobre la componente *Y* afecte a los colores de la imagen.

2.2.1.4 Modelo de color YUV

Este modelo fue definido a partir del modelo *XYZ* ya que éste presentaba una falta de correlación entre las diferencias de color, de acuerdo con la percepción del ojo humano, y la distancia en el espacio *XYZ*. Por ello se definió el sistema UCS (*Uniform Chromaticy Scale*) cuyas coordinas son *Y*, *U*, *V* y vienen dadas por las siguientes expresiones:

$$U = \frac{4X}{X + 15Y + 3Z}$$

$$V = \frac{6Y}{X + 15Y + 3Z}$$

La componente *Y* es la componente de luminancia del sistema *XYZ*. Una ventaja que ofrece este modelo es que las componentes *U* y *V* son ortogonales entre sí, lo que puede ser muy bueno a la hora de usarlo para algoritmos de compresión.

2.2.1.5 Modelo de color HSI

Las siglas HSI corresponden a H: *Hue (Tono)*, S: *Saturation (Saturación)* e I: *Intensity (Intensidad)*. La utilidad del modelo HSI está basado en dos hechos básicos: primero, la componente de intensidad está desacoplada de la información de color y lo que eso lleva consigo de ventajas en el tratamiento de imágenes como hemos descrito anteriormente y segundo, el tono y la saturación están ligados a cómo los humanos percibimos los colores. Por estos dos motivos este modelo de color es ideal para desarrollar algoritmos de procesamiento de imágenes.

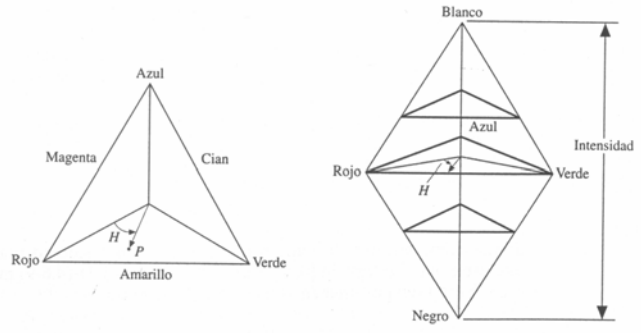


Figura 2-9 : Triángulo de color HSI y sólido de color HSI

La transformación del modelo de color RGB en el modelo de color HSI se realiza en dos pasos, primero las coordenadas RGB se rotan para formar el sistema de coordenadas (I, V_1, V_2). La rotación se describe mediante las ecuaciones lineales que siguen

$$\begin{bmatrix} I \\ V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 \\ 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

a continuación V_1 y V_2 se transforman a coordenadas polares de acuerdo con:

$$H = \tan^{-1}(V_2/V_1)$$
$$S = \sqrt{V_1^2 + V_2^2}$$

CAPÍTULO 3
TEORÍA DE LA INFORMACIÓN

3.1 AUTOINFORMACIÓN Y ENTROPÍA

La respuesta a la pregunta de cuál es la cantidad mínima de datos necesarios para representar una determinada cantidad de información, la tenemos en la Teoría de la Información. Dicha teoría crea un marco matemático en el cual se puede calcular la cantidad de información que posee una imagen.

La premisa fundamental que establece esta teoría, es que la generación de los símbolos de una fuente se pueden modelar como un proceso probabilístico, y que la información que contienen los símbolos se puede medir a través de una fórmula, que se obtiene a través de la intuición. En el caso de la imagen los valores de los píxeles los consideraremos como si se tratara de una fuente de símbolos. [Haykin, 88]

Para un suceso aleatorio *E* con probabilidad *P(E)* necesitamos

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

I unidades de información para representarlo, a este valor se le conoce con el nombre de *autoinformación* o *información propia*. Si no estamos sólo considerando un símbolo, si estamos considerando un conjunto de símbolos que salen de una fuente, -en nuestro caso se tratará de los valores de los píxeles de la imagen- el nombre que se le da a dicha medida de la información de la fuente de datos es *entropía*. Si suponemos una fuente de información con N símbolos, cuyas probabilidades de aparición de cada uno de ellos es $p_{i,}$ y siendo los símbolos estadísticamente independientes la cantidad de información que posee la fuente es

$$H = -\sum_{i=1}^{N} p_i \log(p_i)$$

Dependiendo del tipo de logaritmo, estamos hablando de una medida de información o de otra. El caso que nos interesa es la medida de la información digital, es decir, el bit. Por lo tanto el logaritmo de la fórmula será en base 2. Como se puede observar, para símbolos equiprobables la entropía es mínima.

3.2 TEORÍA DE LA CODIFICACIÓN SIN RUIDO. PRIMER TEOREMA DE SHANNON.

El primer teorema de Shannon define la mínima longitud media de palabra de código por símbolo de fuente que puede obtenerse codificando los símbolos de forma óptima. La longitud mínima de la que hablamos es la entropía. Enunciando formalmente el teorema de Shannon dice así: sea una fuente de información que genera símbolos a_k estadísticamente independientes, sea A el conjunto de todos los símbolos posibles de la fuente, y cada uno de ellos con probabilidades p_k, decimos que la longitud media de palabra de código cumple

$$H(z) < L_{med} < H(z) + 1,$$

Lo que indica es hasta dónde puedo llegar comprimiendo la imagen, ya que si la entropía se ha calculado correctamente, dicho límite no se puede superar por ningún código. Existen limitaciones a lo dicho anteriormente que surgen cuando hablamos de aplicaciones prácticas, estas son:

- Hemos supuesto que los símbolos son estadísticamente independientes, pero en una imagen no ocurre así nunca, porque siempre hay una relación entre los datos de la imagen. Para ello hay que definir una nueva entropía que tiene en cuenta que la información que aporta un símbolo a_i depende de los símbolo que han ocurrido anteriormente, es decir, una probabilidad conjunta, lo que nos lleva a nombrar a esta nueva entropía como *entropía conjunta*.

$$H = -\sum_{a_1,a_2..,a_N} p(a_1,a_2,..,a_N)\cdot \log(p(a_1,a_2,..,a_N))$$

- La entropía no sirve cuando estamos hablando de algoritmos de compresión con pérdidas, ya que el teorema de Shannon se enmarca en la transmisión sin ruido de una información dada, no de una información que ha sido transformada, pero si que la entropía de la imagen transformada por algún algoritmo como el DCT será un límite inferior para la longitud media de la palabra de código.

3.3 ESTIMACIÓN DE LA ENTROPIA DE UNA IMAGEN

En la práctica para calcular la entropía de una imagen necesito conocer las probabilidades de aparición de los símbolos. Lo que se hace es una estimación de las probabilidades, y existen varias estimaciones que exponemos a continuación. Vamos a basarnos en la imagen de 3x3 de la Figura 3-1 para ejemplificar el uso de una u otra estimación.

21	95	169
21	95	169
21	95	143

Figura 3-1 : Matriz que representa a los valores de una imagen.

3.3.1 Estimación de Primer Orden

Supongo que el modelo de fuente es sólo mi imagen, pero en realidad necesitaría más imágenes del mismo tipo. Esta estimación es lo mismo que hacer el histograma a un conjunto de datos. En la imagen calculamos la probabilidad de aparición de un nivel de gris.

Nivel de Gris	*Número de apariciones*	*Probabilidad*
21	3	3/9
95	3	3/9
169	2	3/9
143	1	1/9

Figura 3-2 : Probabilidades con estimación de primer orden de los distintos niveles de gris de la imagen

3.3.2 Estimación de Segundo Orden

En este caso se calcula la frecuencia de aparición de bloques de píxeles, donde un bloque son dos píxeles continuos. La imagen se lee de izquierda a derecha y se une el final de la una fila con el principio de la siguiente.

Nivel de Gris	*Número de apariciones*	*Probabilidad*
(21,95)	3	3/9
(95,169)	2	2/9
(169,21)	1	1/9
(95,143)	1	1/9
(143,21)	1	1/9

Figura 3-3 : Probabilidades con estimación de segundo orden de los distintos niveles de gris de la imagen

3.3.3 Estimaciones de orden superior

A medida que sube el orden mejora la estimación de la entropía, pero requiere de mucho cálculos convergiendo al final al valor de la entropía. Se suelen usar los estimadores de primer y segundo orden que hemos vistos anteriormente por su menor coste computacional.

CAPÍTULO 4

FUNDAMENTOS DE COMPRESIÓN DE IMÁGENES

4.1 INTRODUCCIÓN

Lo primero que deberíamos hacer es diferenciar dos conceptos que pueden llegar a parecer lo mismo pero no lo son. Tenemos que diferenciar entre información y datos. La información se representa por datos y una misma información puede ser representada por diferentes conjuntos de datos. El hecho de que se pueda representar una determinada información por diferentes conjuntos de datos, hace que haya unos conjuntos de datos que ocupen un espacio físico, por ejemplo en una memoria, mayor que otros. Por lo tanto habrá algunos datos que serán redundantes porque si existe otro conjunto con un número menor de datos se podría mejorar el conjunto redundante. La redundancia no es un concepto abstracto sino un concepto que se puede medir matemáticamente. Si tenemos dos cantidades de unidades de información n_1 y n_2 que representan el mismo número de unidades de información, la *redundancia relativa de los datos* R_D se define como:

$$R_D = 1 - \frac{1}{C_R}$$

donde C_R se llama *relación de compresión*

$$C_R = \frac{n_1}{n_2}$$

el significado de estas expresiones puede observarse si tenemos en cuenta tres casos particulares. Primero, en el caso que $n_1 = n_2$ nos da un valor de $C_R = 1$ y $R_D \to 0$ quiere decir que el segundo conjunto no tiene redundancia con respecto al primero. Segundo,

que $n_2 << n_1$ entonces $C_R \rightarrow \infty$ y $R_D \rightarrow 1$, quiere decir que el número de unidades de datos del segundo conjunto es mucho menor que el del primero, por lo tanto hay una compresión importante de datos de un conjunto a otro, de ahí que la relación de compresión tienda a infinito. Por último se puede dar el tercer caso particular de que $n_1 << n_2$, $C_R \rightarrow 0$ $R_D \rightarrow -\infty$, esto indica que el segundo conjunto de datos posee una gran redundancia de datos porque ocupa muchas más unidades de datos que el primero, por eso la redundancia tiende a infinito.

En un esquema de compresión se han de desarrollar algoritmos que hagan que la relación de compresión sea la más alta posible, que es lo mismo que decir que el segundo conjunto de datos que obtengamos después de aplicar el algoritmo tengan menos redundancia y su tamaño sea menor para representar la misma cantidad de información.

En la compresión de imágenes se pueden aprovechar cuatro tipos de redundancias: la redundancia intrapíxel, la redundancia interpíxel, la redundancia de codificación y la redundancia psicovisual. Se habrá conseguido una compresión cuando se reduzcan o se eliminen alguna de estas redundancias.[González, 96]

4.2 MÉTODOS GENERALES DE COMPRESIÓN

El esquema general de compresión que podemos ver en cualquier sistema de compresión se basa en el de la Figura 4-1, en el que puede que algunos bloques estén presentes o no, dependiendo del tipo de compresión, por ejemplo si esta tiene o no pérdidas.

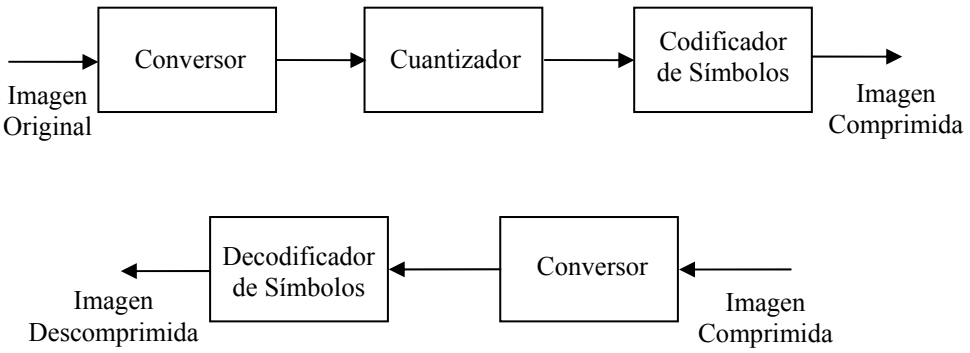


Figura 4-1 : Esquema general de compresión y descompresión

Podemos clasificar los métodos de compresión en dos grandes conjuntos, que son la compresión de imágenes sin pérdidas o la compresión con pérdidas:

- En la *compresión sin pérdidas* si tenemos una imagen que queremos comprimir para posteriormente transmitirla y descomprimirla en el receptor, en el otro lado de la comunicación, esto es, el receptor, tenemos exactamente la misma imagen que en el transmisor, cada píxel tendrá los tres mismos niveles que tenía en su origen. Este tipo de compresión se usa sobre todo en casos en que la información de la imagen es tan importante o ha sido tan cara obtenerla que una pérdida en la compresión no se puede permitir, estamos hablando de

proyectos espaciales, los de alto valor capital. También los documentos legales así como imágenes médicas en las que los detalles, que pueden llevar a una detección por ejemplo en una radiografía de una enfermedad, son tan importantes que no deben perderse en el proceso de compresión.

- En la *compresión con pérdidas* la imagen que tenemos en el receptor no tiene exactamente los mismos valores de los píxeles de la que hemos enviado. En estos algoritmos la importancia reside en que al descomprimir se perciba en conjunto lo que en el transmisor ha comprimido, es decir, que lo que se busca no es que los datos sean los mismos, sino que la percepción por el sistema visual humano sea muy parecido. Con la compresión con pérdidas se consiguen tasas de compresión más altas que los algoritmos sin pérdidas. Sobre todo se usan estos algoritmos para videoconferencia, fax, intercambiar fotos…

El cuantizador es un elemento que normalmente determina si un algoritmo posee pérdidas o no, podemos decir que si un esquema de compresión posee un cuantizador, éste tendrá pérdidas, pero también puede darse el caso que sin tenerlo en el proceso de conversión se haga una transformación de la imagen que conlleve la introducción de pérdidas en el esquema. Por ello a veces se suele emplear transformadas de la imagen en compresión con pérdidas, y métodos basados en la predicción en compresión sin pérdidas.

A continuación vamos a describir los principales tipos de redundancias y los métodos que normalmente se usan para eliminarlas y así comprimir la imagen.

4.2.1 Redundancia interpíxel

Este tipo de redundancia está directamente relacionada con las correlaciones entre los píxeles de una imagen. Es posible predecir de forma razonable un píxel a partir de los vecinos, lo que indica que la información que posee un píxel de forma individual es pequeña. La contribución que ofrece un píxel a una imagen completa es muy pequeña, y en el entorno de sus vecinos es redundante, porque si dicho píxel no existiera se percibiría la imagen de casi la misma forma que si el píxel estuviera. El valor de dicho píxel podría haberse inferido de acuerdo con los valores de los vecinos. A este tipo de redundancia en que el valor de un píxel en un conjunto sea redundante se le denomina redundancia interpíxel y existen métodos para reducirla o eliminarla.

4.2.1.1 Métodos de eliminación de la redundancia interpíxel

Según los métodos de eliminación de la redundancia interpíxel podemos clasificar los esquemas de compresión en:

- *Compresión basada en la transformada de la imagen*: consiste en cambiar las coordenadas de representación de la imagen a otro dominio transformado, en el que los coeficientes transformados estén menos correlacionados que en el dominio original.
  La decorrelación total de los coeficientes transformados es equivalente a la diagonalización de la matriz de autocovarianza, ésta se consigue con la transformada Karhunen-Loève, aunque otras tan populares como la DCT

(transformada del coseno) y las transformadas wavelet o subbanda debidamente diseñadas, pueden llegar a la diagonalización.

- *Compresión predictiva*: consiste en aprovechar la correlación existente entre los píxeles para predecir un píxel a partir de los anteriores y almacenar sólo la diferencia, con lo cual el rango dinámico de los datos será menor y el número de bits necesarios para su representación será también menor.

4.2.1.1.1 Basados en la transformada de la imagen

Se aplica la transformación a la imagen para llevarla a otro dominio de representación transformado, que debe cumplir que los coeficientes transformados estén menos correlacionados entre sí. La idea es la de compactar la energía de la imagen.

Por comprobación práctica se demuestra que la cuantificación escalar de los coeficientes resultantes tras la decorrelación, es más eficiente que la cuantificación de los coeficientes escalar directa de las muestras. La propiedad de la compactación de la energía consiste en que la mayoría de la energía de la señal se encuentra concentrada en una parte de los coeficientes transformados, eso quiere decir que se pueden despreciar un gran número de coeficientes, lo que conlleva una compresión. Esta compresión se da en la cuantificación y no en la transformación.

Debido a la no estacionalidad de la imagen, la imagen se divide en bloques a los cuales se les aplica la transformación, para conseguir así que todos los bloques sean aproximadamente estacionarios.

4.2.1.1.1.1 Transformadas lineales

Una transformación lineal aplicada sobre una imagen es aquella transformación que, puede expresarse como:

$$T_f(k_1,k_2) = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} f(n_1,n_2)\cdot a(n_1,n_2;k_1,k_2)$$

$$f(n_1,n_2) = \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} T_f(k_1,k_2)\cdot b(n_1,n_2;k_1,k_2)$$

donde $f(n_1,n_2)$ es una imagen N_1xN_2, $T_f(k_1,k_2)$ es una secuencia también N_1xN_2 que representan los coeficientes transformados, y $a(n_1,n_2;\ k_1,k_2)$ y $b(n_1,n_2;\ k_1,k_2)$ son las funciones base. Las transformadas que se han estudiado más profundamente has sido la transformada de Harhunen-Loève (KLT), la transformada de Fourier discreta (DFT) y la transformada discreta del coseno (DCT). Todos tienen comportamientos similares, difiriendo entre ellas la compactación de la energía que alcanzan cada una de ellas y el coste computacional que llevan consigo cada una.

1) Transformada de Karhunen-Loève

Es la mejor transformación lineal en cuanto a la compactación de la energía. En la *KLT* las funciones bases se obtienen, salvo un factor de escala (λ), de la ecuación:

$$\lambda(k_1,k_2)\cdot a(n_1,n_2;k_1,k_2)=\sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1}K_f(n_1,n_2;l_1,l_2)\cdot a(l_1,l_2;k_1,k_2)$$

$$K_f(n_1,n_2;l_1,l_2)=E\left[(f(n_1,n_2)-E\left[f(n_1,n_2)\right])(f(l_1,l_2)-E\left[f(l_1,l_2)\right])\right]$$

en la primera ecuación se supone que $f(n_1,n_2)$ es un proceso aleatorio de función de covarianza conocida e igual a $K_f(n_1,n_2;\ l_1,l_2)$. En tanto que esta suposición es válida, tras la transformación se ha eliminado completamente la correlación entre los píxeles de la imagen. Bajo esta suposición la compactación de la energía es máxima.

A pesar de los excelentes resultados teóricos que presenta dicha transformada, posee serias dificultades a la hora de ponerla en práctica, ya que no existen algoritmos computacionalmente eficientes, y por otra parte la KLT se basa en la suposición de que la imagen es un proceso aleatorio de covarianza conocida. En la práctica, dicha función no es válida, y la función de covarianza debe de ser estimada.

2) Transformada DFT

La DFT y la versión del algoritmo de cálculo de forma eficiente FFT, poseen un conjunto de funciones base que consiguen una buena compactación de la energía. La DFT, desde que el procesado digital de señales existe, ha sido usada ampliamente, por ello cuando se empezaron a desarrollar las técnicas de codificación de imagen se empezó a utilizar también. Pero presenta una desventaja, ya que la transformada DFT coincide con los coeficientes de la serie de Fourier de la imagen $f(n_1,n_2)$ de tamaño N_1xN_2 extendida periódicamente, por lo que al extender la imagen se presentan unas discontinuidades en los bordes de la imagen que se traducen en valores altos para los coeficientes en alta frecuencia, lo que empeora la compactación de la energía.

3) *Transformada DCT*

Esta transformada evita el problema que tenía la DFT y conserva la eficiencia en el cálculo. Para evitar las discontinuidades se crea una secuencia $g(n_1,n_2)$ de tamaño $2N_1x2N_2$,que es la secuencia $f(n_1,n_2)$ completada por la derecha con $f(2N_1-n_1,n_2)$, por arriba con $f(n_1,2N_2-n_2)$ y en su diagonal por $f(2N_1-n_1,2N_2-n_2)$. Los coeficientes transformados se calculan como los coeficientes DFT de la secuencia $g(n_1,n_2)$.

4.2.1.1.1.2 Transformadas wavelet o subbanda

La compresión de señales emplea, en la mayoría de los casos, un bloque de expansión de la señal que se lleva a cabo por medio de bancos de filtros. Cuando estos bancos de filtros se usan para codificar, el esquema recibe el nombre de *codificación subbanda* o *técnicas de multiresolución*, porque la expansión de la señal consiste en descomponerla en una señal aproximación más una señal llena de detalles. Si el conjunto de bancos se encuentra en cascada, nos conduce a una sucesión de señales aproximación más las señales con los detalles. Esto es lo que se denomina en las técnica multirresolución como *aproximación sucesiva*. De igual modo que se emplea en la compresión de señales, también se emplea en la compresión de imágenes.

4.2.1.1.2 Basados en predicción

Se basa en la eliminación de la redundancia existente entre píxeles cercanos. Lo que se hace es codificar la información que aporta el píxel al contexto. La información es la diferencia entre el valor real del píxel a codificar y el valor estimado de dicho píxel. Con esto se consigue que el rango dinámico de los píxeles a codificar sea menor.

El esquema de un compresor basado en predicción es el que muestra la Figura 4-2

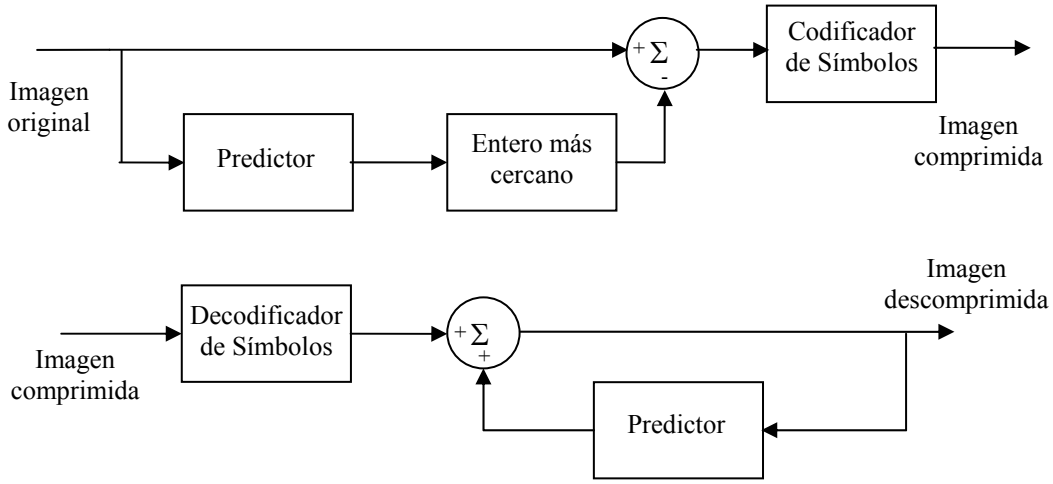


Figura 4-2 : Esquema de un sistema de codificación predictiva sin pérdidas

donde se predice el valor del píxel a codificar basándose en el entorno de éste y se redondea al entero más cercano, y este valor se resta al valor del píxel que será lo que se codificará con el código normalmente de longitud variable.[Gonzalez, 96]

4.2.2 Redundancia *intrapíxel*

Este tipo de redundancia sólo se da en las imágenes a color, ya que los planos de color R, G y B están altamente correlacionados. Si tenemos un algoritmo de compresión funcionando bajo imágenes en niveles de gris, antes de aplicarlo a una imagen a color, habría que reducir esta redundancia para que el algoritmo funcionara de un modo más eficiente.

El método más sencillo para reducir esta redundancia es aplicar a la imagen un cambio del espacio de color a uno de representación más adecuado para la compresión. Se consigue reducir la entropía de la imagen si se transforma a un espacio de color ortogonal. El espacio de color *YIQ* es un ejemplo de ello, posee la componente *Y*: luminancia separada de la crominancia, que esta a su vez se divide en dos componentes ortogonales *I*: in-phase y *Q*: in-quadrature. Este sistema de representación es muy adecuado para la compresión de imágenes porque podemos aplicar el algoritmo independientemente a las tres coordenadas por separado, porque ya hemos disminuido la redundancia que impedía aplicar un algoritmo de escala de grises a una imagen a color sin cambiar las coordenadas de color. Otra ventaja de este sistema es que las componentes de alta frecuencia se tienden a concentrar en la componente *Y,* lo que se traduce, hablando de compresión, en que se le dará un mayor número de coeficientes

para la representación de Y y menos a las otras dos componentes. Otro espacio también ortogonal usado es YUV .

Si estamos hablando de compresión con pérdidas, también se reduce la correlación intrapíxel empleando una cuantificación vectorial en tres dimensiones donde cada una de ellas se corresponde a un plano de color del píxel. Otra ventaja es que de los 16 millones de colores no todos se usan, sólo se usan una pequeña parte. Lo que se suele hacer es crear una paleta con 256 colores y almacenar la imagen en un tercio del espacio que se requería previamente.

4.2.3 Redundancia en la codificación

La redundancia en la codificación viene dada porque los símbolos que van saliendo de la fuente de información, en nuestro caso los valores de los píxeles de la imagen, no todos ellos poseen la misma probabilidad de aparición, por lo que un símbolo que sea poco probable no debería codificarse con el mismo número de bits que un símbolo que aparece continuamente, es decir, el símbolo muy probable debería codificarse con pocos bits y el menos probable con más. Si hacemos una estimación buena de la entropía de la imagen, ésta nos dará el límite inferior, que indicará el límite inferior posible de longitud media de las palabras de código.

Los algoritmos que llevan a cabo la reducción de la redundancia en la codificación se le conoce con el nombre de codificadores entrópicos, ya que la entropía es la que impone el número medio más bajo de bits con los que se puede codificar los símbolos. Normalmente el método más efectivo para crear una codificación eficiente, es una codificación de longitud variable. Con esta variabilidad en la longitud del código se consigue que, la redundancia debida a la asignación de un código binario natural que asignaría el mismo número de bits a cada uno de sus niveles se reduzca.

Si nosotros tenemos una imagen en nivel de gris y hacemos el histograma de dicha imagen

$$p_k = \frac{n_k}{n} \quad k = 0,1,2,....,L-1$$

obtenemos p_k la probabilidad de aparición del valor k, siendo n_k el número de apraciciones del valor k y siendo n el número total de píxeles de la imagen. Según la Tabla 4-1 vemos como los símbolos más probables tiene un código de longitud menor que los símbolos menos probables.

p_k	*Código 1*	L_1	*Código 2*	L_2
0.19	000	3	11	2
0.25	001	3	01	2
0.21	010	3	10	2
0.16	011	3	001	3
0.08	100	3	0001	4
0.06	101	3	00001	5
0.03	110	3	000001	6
0.02	111	3	000000	6

Tabla 4-1: Ejemplo de código de longitud variable

Si calculamos la longitud media de ambos códigos mediante la fórmula:

$$L_{med} = \sum_{k=0}^{L-1} l(r_k) \cdot p_k$$

observamos que para el primer código la longitud media es de 3bits, ya que todos ellos tienen longitud 3. Para el segundo código, al que hemos aplicado una codificación de longitud variable dependiendo de la probabilidad de aparición de los símbolos, hemos obtenido el resultado de la longitud media:

$$L_{med_2} = 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) + \\ + 5(0.06) + 6(0.03) + 6(0.02) = 2.7\,bits$$

Como vemos hemos obtenido una reducción en la longitud de la palabra de código de 0.3bits con una simple codificación variable, que a pesar que puede parecer poco, para ver su importancia habría que multiplicar el tamaño de la imagen *NxM* por la L_{med} y la diferencia aparecerá significativa.

Los dos codificadores entrópicos por excelencia son el *codificador de Huffman* y el *codificador aritmético*. Existen en la literatura otros codificadores que tratan de mejorar los resultados de éstos, pero no son más que modificación de estos dos.

4.2.3.1 El Codificador de Huffman

Es la técnica más popular de eliminación de la redundancia de la codificación. Cuando se codifican individualmente los símbolos de una fuente, el codificador de Huffman consigue el número más pequeño posible de símbolos de código por símbolo de fuente. La única restricción que posee este codificador es que los símbolos de la fuente se deben codificar uno a uno.

Este algoritmo se puede explicar en dos pasos o en uno, nosotros vamos a explicarlo en dos pasos ya que así se comprende su funcionamiento mejor. En un primer paso se calculan las probabilidades de los símbolos de la fuente y se ordenan de mayor a menor según el valor de la probabilidad de aparición. Se va procediendo a sucesivas

reducciones de fuente en cada una de las cuales se suman los valores de menor probabilidad y se colocan en la siguiente reducción de fuente en el lugar correspondiente dependiendo del valor de la suma, es decir, que en cada reducción de fuente las probabilidades deben de estar ordenadas. Se procede de este modo hasta que sólo tenemos dos probabilidades y es el momento en que el que se pasa a la segunda parte del procedimiento.

Fuente Original		*Reducción de Fuente*			
Símbolo	Probabilidad	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Figura 4-3 : Reducciones de una fuente, según el método de Huffman

En una segunda etapa se codifica cada fuente reducida empezando por la más pequeña, hasta llegar a la fuente original. En la última reducción de fuente se le asignan 0 y 1 y según se haya reducido la fuente así debe crecer, esto es, si el valor de 0.6 se subdivide en los dos de 0.3 como se ve en la Figura 4-4, a cada uno de ellos se le asigna el 00 y el 01. A continuación se ve cual de los tres fue generado en la reducción de la fuente, el cual se divide a su vez en los dos siguientes empezando por el código del que provienen, en este caso 010 y 011. Así se continúa hasta que se termina el proceso inverso de la reducción de fuente. El resultado que obtenemos es el siguiente con su correspondiente código.

Fuente Original			*Reducción de Fuente*							
Símb.	Prob.	Código	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Figura 4-4 : Procedimiento para la asignación de códigos.

Si calculamos la longitud media del código:

$$L_{med} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + \\ + (0.04)(5) = 2.2bits / símbolo$$

si la entropía de la fuente es 2.14 bits/símbolo obtenemos una muy alta eficiencia de código, 0.973.

Después de crear el código, la codificación y la decodificación se hace simplemente accediendo a la tabla. Es un *código bloque descodificable instantáneamente de manera única*. Código bloque porque a cada símbolo de la fuente se le hace corresponder con una secuencia fija, es instantáneo ya que cada palabra de código se puede decodificar sin tener en cuenta los demás símbolos, y es descodificable de manera única, porque una cadena de símbolos de códigos sólo se puede decodificar de una forma.

4.2.3.2 Codificador aritmético

El algoritmo de codificación aritmética no genera códigos de bloque como lo hace el algoritmo de Huffman, por el cual cada símbolo de la fuente le corresponde una palabra de símbolo. En este codificador no existe una correspondencia biunívoca entre los símbolos de la fuente y las palabras de código. Se asigna una única palabra de código a una secuencia de símbolos de la fuente. La propia palabra de código define un intervalo de números reales entre 0 y 1. Conforme aumenta el número de símbolos del mensaje, el intervalo utilizado para representarlo se va haciendo menor y se va incrementando el número de unidades de información necesarias para representarlo. Cada símbolo que aparece reduce el tamaño de los intervalos según su probabilidad de aparición.

Supongamos el ejemplo de la Figura 4-5 para explicar el funcionamiento del codificador aritmético, para lo cual vamos a tener una fuente de información de la cual pueden surgir cuatro símbolos diferentes cada uno de ellos con una probabilidad de aparición.

Símbolo fuente	*Probabilidad*	*Subintervalo inicial*
a_1	0.2	[0.0; 0.2)
a_2	0.2	[0.2; 0.4)
a_3	0.4	[0.4; 0.8)
a_4	0.2	[0.8; 1.0)

Figura 4-5 : Probabilidad de los símbolos del ejemplo

Se desea codificar una secuencia de cinco símbolos $a_1a_2a_3a_3a_4$, al principio del proceso el mensaje ocupa todo el espacio [0,1). Inicialmente este intervalo está dividido en cuatro regiones, dependiendo su tamaño de la probabilidad. Como el primer símbolo es a_1 el intervalo del mensaje se reduce al de este símbolo [0, 0.2), el siguiente símbolo es a_2 y el intervalo de éste será la parte del intervalo [0, 0.2) que le corresponde por proporción dependiendo de la probabilidad que tuviera. Así se continúa hasta el último

que se reduce el intervalo a [0.06752; 0.688), cualquier número de este intervalo será válido para representar este mensaje.

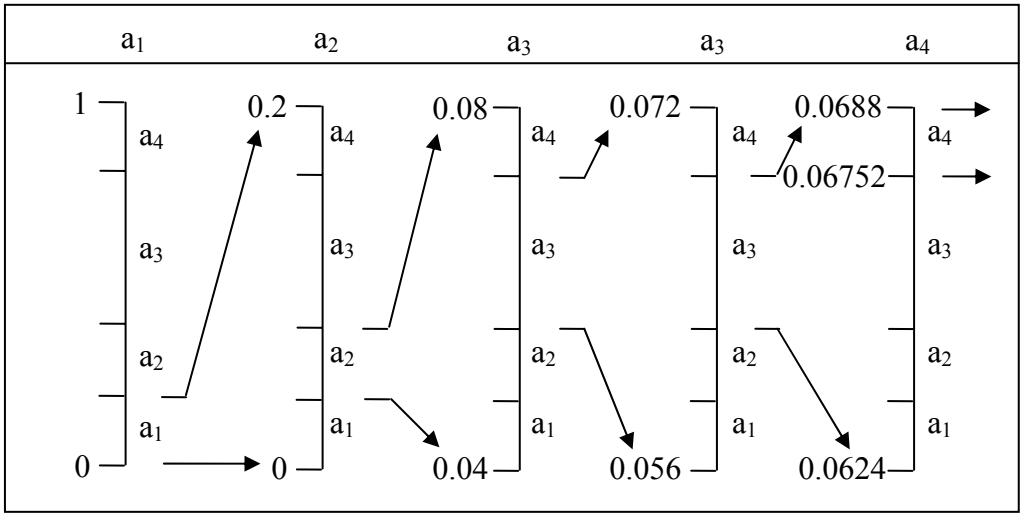


Figura 4-6 : Proceso de codificación aritmética

En la práctica, existen dos factores que hacen que el rendimiento de la codificación se aleje del límite teórico de la entropía, uno es que se debería incluir un indicador de fin de mensaje para separar un mensaje de otro, y el segundo es que es necesaria la utilización de aritmética entera. Las implementaciones prácticas de la codificación aritmética solucionan este último problema utilizando una estrategia de escalado y otra de redondeo.

4.2.4 Redundancia psicovisual

Siempre que un sistema de compresión presente un método de reducción de la redundancia psicovisual, estaremos hablando de un sistema de compresión con pérdidas. Esto quiere decir que la imagen descomprimida no es exactamente la misma que la original. Con métodos con pérdidas se alcanzan niveles de compresión más altos que los esquemas con pérdidas. La redundancia psicovisual se basa entre otros muchos factores, que el sistema visual humano no es capaz de distinguir todos los colores o niveles que se pueden representar en una imagen y tampoco de percibir una disminución de la resolución espacial, entre otras propiedades.

El buen funcionamiento de un método de reducción de redundancia psicovisual depende de un bloque anterior, el conversor, y cómo éste ha sido capaz de compactar la energía de la imagen, puesto que si algunos coeficientes transformados son 0, no se codificarán. Para que la cuantificación de la imagen transformada sea eficiente, se necesita el diseño de un cuantizador óptimo para la imagen a codificar. Para conseguir un cuantizador óptimo es necesario conocer la función de densidad de probabilidad de la distribución de símbolos a la entrada.

Si llamamos {a(n,m)} a la secuencia de símbolos que constituyen la imagen, {$\hat{a}_i$} al conjunto de niveles de cuantificación y f(â-a) a la función de distorsión, entonces el resultado de la distorsión D es :

$$D = \sum_{M} f(\hat{a} - a)\, p(a)$$

donde M es el número de posibles símbolos de entrada y p(a) es la probabilidad de ocurrencia del símbolo a.

Un cuantificador óptimo es aquel que minimiza D mediante la selección de los niveles de salida y los rangos de entrada correspondientes a cada nivel de salida. Este problema está optimizado por las investigaciones de Lloyd y Max de ahí el nombre del cuantificador, *cuantificador de Lloyd y Max*.

CAPÍTULO 5

REVISIÓN DE ALGUNOS ALGORITMOS DE COMPRESIÓN SIN PÉRDIDAS

En este capítulo se va a realizar una revisión de algunos de los algoritmos de compresión sin pérdidas. Alguno de los conceptos de los siguientes compresores son usados en los algoritmos desarrollados posteriormente. Los algoritmos que se van a revisar son:

1. JPEG-LS: Estándar para la compresión de imágenes sin pérdidas.
2. Estándar JBIG.
3. Algoritmo SLIC.

5.1 JPEG-LS : ESTÁNDAR PARA LA COMPRESIÓN DE IMÁGENES SIN PÉRDIDAS.

El algoritmo JPEG-LS es un algoritmo de compresión de imágenes sin pérdidas. Surgió por la necesidad de mejorar la anterior versión, el JPEG, que estaba siendo superado ampliamente por la aparición de nuevos algoritmos. Es una algoritmo estandarizado; el primer documento con relación a este estándar se publicó en 1996 [ISO, 96] y posteriormente surgió la versión definitiva. Se desarrolló buscando una alta compresión usando un algoritmo de muy poca complejidad. El algoritmo base del JPEG-LS es el *LOCO-I* (**LO**w **CO**mplexity **LO**ssless **CO**mpression for **I**mages). El algoritmo JPEG-LS posee un alto nivel de compresión utilizando una menor complejidad computacional. [Weinberger, 99][Weinberger, 00]

El algoritmo JPEG-LS se basa en:

- Codificación predictiva (no usa transformadas)
- Modelos de predicción y codificación simples
- Codificación con códigos de Golomb elegidos adaptativamente
- Procesamiento especial para regiones de baja entropía

El algoritmo utiliza un modelo de predicción y un modelo de codificación. Con el fin de reducir el costo computacional del modelo, la predicción está basada en la misma información que se utiliza para construir el modelo de codificación.

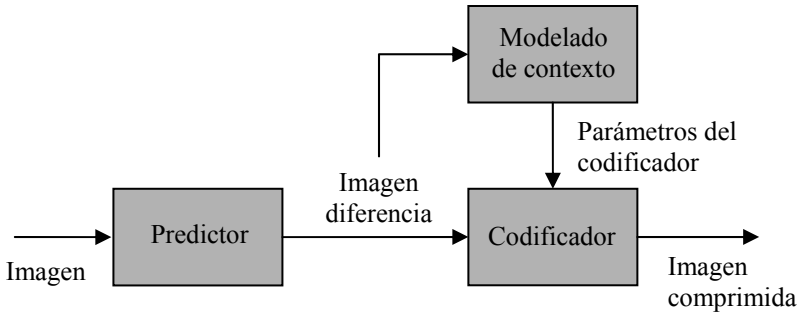


Figura 5-1 : Esquema básico de algoritmo JPEG-LS

En términos generales el algoritmo de puede dividir en tres bloque importantes:

- *Predictor*
- *Modelado de contexto*
- *Codificador*

5.1.1 Predictor

El modelo de predicción del JPEG-LS establece que el píxel en cuestión que estemos tratando en cada momento *x*, depende de los valores de los píxeles previos *a*, *b, c* y *d*. La posición que tienen los valores *a*, *b* ,*c* y *d* son los de la Figura 5-2.

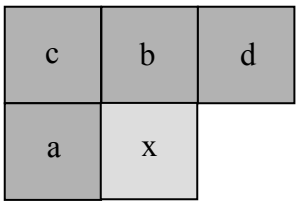


Figura 5-2 : Esquema de predicción en el algoritmo JPEG-LS

El predictor está compuesto de una componente fija que es un filtro de mediana y otra componente adaptativa que realiza una cancelación de offset . Como ésta última tiene

relación con el modelado del contexto se verá en el siguiente apartado. La componente fija del predictor MED (Median Edge Detector) posee una capacidad rudimentaria de detección de bordes. Es una función no lineal que actúa sobre *a*, *b* y *c* como vemos en la ecuación que rige:

$$x_{med} = \begin{cases} \min(a,b) & si \;\; c > \max(a,b) \\ \max(a,b) & si \;\; c \le \min(a,b) \\ a+b-c & en \; otro \; caso \end{cases}$$

Se puede ver cómo el valor de *x* tiende a *b* si detecta un borde vertical, y tenderá a *a* si detecta un borde horizontal. Si por el contrario no detecta ningún borde, ni horizontal ni vertical, el valor de *x* tiende a *a*+*b*-*c*. Otra forma de ver este predictor es, que escoge el valor predicho como el valor mediano de tres predoctores independientes como son *a*, *b* y *a*+*b*-*c*.

5.1.2 Modelado del contexto

En el modelo del contexto se parte de que la distribución estadística de los residuos de un esquema de predicción fija de imágenes se puede modelar por una distribución geométrica lateral centrada en 0. La probabilidad de que un residuo tenga un valor entero ε es proporcional al valor $\theta^{|\varepsilon|}$, donde θ pertenece al intervalo (0,1). Se observó que presentaba un valor de continua en los errores de predicción. Esto se debe a la restricción de valores enteros. Por lo tanto se hace necesario introducir un parámetro adicional μ, que representa el valor de continua, que a su vez se descompone en una suma R+s donde R es la parte entera y s es la parte fraccionaria. De este modo la distribución de los residuos del predictor queda como:

$$P_{(\theta,\mu)}(\varepsilon) = C(\theta,\mu)\cdot\theta^{|\varepsilon-\mu|}$$

donde el valor de $C(\theta,\mu)$ viene dado por:

$$C(\theta,\mu) = \frac{1-\theta}{\theta^{1-s}+\theta^{s}}$$

que es un factor de normalización. La parte adaptativa del predictor calcula un valor que es el offset con lo que el valor entero R a partir de ahora se omitirá porque ha sido cancelado por la parte adaptativa del predictor. Los demás parámetros de la distribución son dependientes del contexto.

En el cálculo del contexto intervienen los gradientes locales:

$$g_1 = d - b$$
$$g_2 = b - c$$
$$g_3 = c - a$$

con estos tres gradientes se puede conocer el nivel de actividad de la zona del píxel que estamos teniendo en consideración. Los tres gradientes influyen de la misma manera en

el algoritmo, es decir, los tres tienen el mismo peso. Se cuantifican los gradientes en regiones (intervalos) equiprobables. Estos intervalos son fijos pues las limitaciones de complejidad no permiten división en regiones adaptada a la imagen.

La división en intervalos debe cumplir dos condiciones:

- Tener a {0} como una de las regiones.
- Ser simétrica respecto a cero.

Las regiones se indexan así { -T,…,-1,0,1,…,T}. Por simetría se puede suponer que la probabilidad :

$$\Pr ob\{\varepsilon_t = \delta / C_t = [q_1, q_2, q_3]\} = \Pr ob\{\varepsilon_t = -\delta / C_t = [-q_1, -q_2, -q_3]\}$$

siendo C_t el contexto calculado para el píxel anterior al que se trata de predecir, y las tres componentes q_1,q_2,q_3 son las tres componentes g_1,g_2,g_3 calculadas. Por lo que las $(2T+1)^3$ posibles combinaciones de la triada $\{g_1, g_2, g_3)$ se pueden reducir teniendo en cuenta lo anterior a

$$\frac{(2T+1)^3 + 1}{2}$$

si se agrupan en uno solo los contextos numerados como $\pm k$.

En el algoritmo de JPEG-LS se toma el valor de T=4 con lo que se obtienen un número de 365 contextos. Las regiones de cuantificación que se definen por defecto son: {0},±{1,2},±{3,4,5,6},±{7,8,…,20},±{ε/ε ≥ 21}. Los límites son variables pero la región {0} debe conservarse.

Esta elección de los 365 contextos ha sido una elección de compromiso entre la capacidad de compresión del algoritmo y el costo computacional. Conforme aumente el número de contextos la compresión será mayor, pero hay que decidir si, subiendo ese número y observando el aumento de compresión, es factible considerando que el tiempo de compresión aumenta.

5.1.3 Codificador

El codificador usado en el algoritmo JPEG-LS tiene mucho que ver con la distribución probabilística de los errores de predicción. Como ya se dijo, estos errores siguen una distribución geométrica bilateral. De ahí que se emplee un codificador que sea óptimo para ese tipo de distribuciones. El codificador que se usa se basa en los códigos de *Golomb*, que son óptimos para distribuciones geométricas unilaterales.

5.1.3.1 Códigos de Golomb

Un código de *Golomb* de orden *m* codifica un número entero $y \geq 0$ como una concatenación de una:

- representación unitaria de $\lfloor y/m \rfloor$
- representación binaria de $y \bmod m$

Un caso particular de código de Golomb es cuando m = 2^k, es decir, el valor *m* es potencia de dos y se representa por *GPO2* y un número se codifica por:

- representación unitaria del número formado por los (n-k) bits más significativos
- y otra representación formada por los k bits menos significativos.

Un ejemplo para ello es el siguiente: si tenemos un código de Golomb con m=8=2^3 y queremos representar el número y=20 será como sigue:

$$y = 20 \quad \Rightarrow y_b = 10100 \quad \Rightarrow G(y) = 110100$$

5.1.3.2 Codificación particular de JPEG-LS

Para el caso particular de la codificación JPEG-LS se utiliza una subfamilia de los códigos de Golomb que son los GPO2, es decir, aquellos en los que m es potencia de dos. Para cada θ, siendo este el parámetro que caracteriza el ancho de la distribución, existe un valor *m* tal que G_m proporciona la longitud media de palabra de código mas corta unívocamente descifrable.

El espacio generado por el par de parámetros *(θ,s)* se divide en regiones y se asigna un código diferente a cada región. El código de Golomb tendrá en cada región un valor distinto de *m* en función de los parámetros *(θ,s)*.

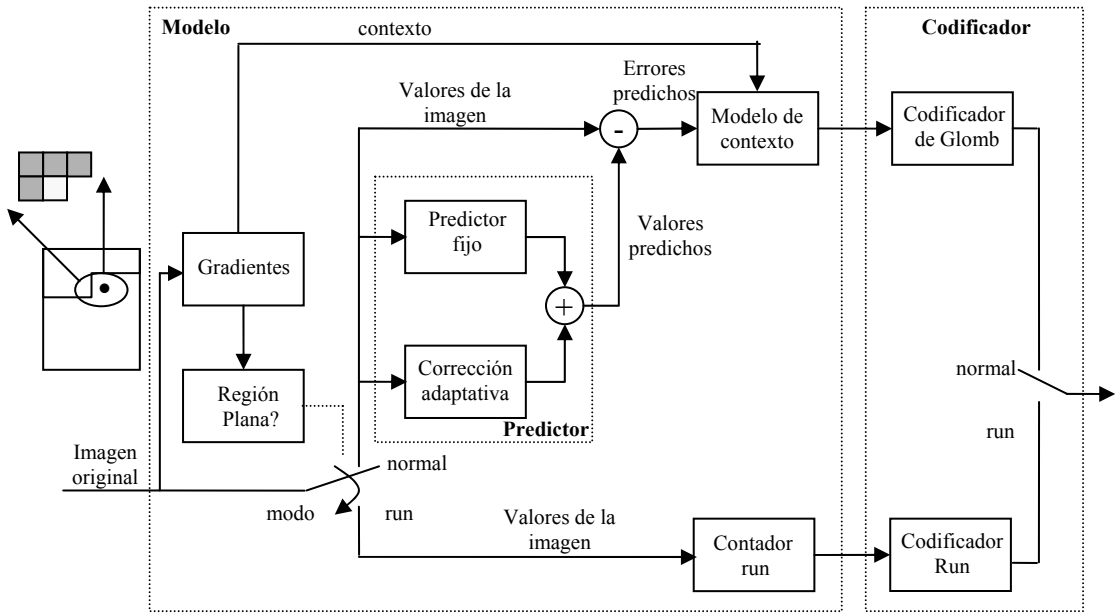


Figura 5-3 : Esquema de compresión del estándar JPEG-LS

5.1.4 Modos de funcionamiento

Además del modo normal de funcionamiento descrito arriba para el algoritmo JPEG-LS, éste posee otro modo de funcionamiento exclusivamente para las regiones de baja entropía. Al modo se le denomina *RUN* y en las regiones con baja entropía donde no es eficiente la codificación símbolo a símbolo se aplica este método.

El codificador entra en funcionamiento *RUN* sí $g_1 = g_2 = g_3 = 0$ ($[q_1, q_2, q_3] = [0,0,0]$), se espera una ristra del símbolo *a* cuyo largo se codifica utilizando códigos de Golomb, y se sale del modo *RUN* cuando deja de repetirse el símbolo *a* o por el final de línea. Cuando el algoritmo está en modo *RUN* no se chequea contexto, ni se realiza predicción.

5.2 ESTANDAR JBIG

El formato JBIG es un estándar para la codificación de imágenes de dos niveles, esto es, que cada píxel se representa por 1bit. Dicho formato es estándar de la CCITT del año 1993. Este algoritmo tiene altas tasas de compresión incluso a veces superiores al JPEG sin pérdidas, pero a pesar de ello, no es un estándar de gran popularidad debido a que es una patente de IBM. [UIT-T, 93]

Es un método que preserva los bits en el proceso de compresión-descompresión por lo que, como debería ser por estar enmarcado en este capítulo, estamos hablando de un algoritmo de compresión sin pérdidas.

El codificador puede descomponerse como se muestra en la Figura 5-4 por medio de bloques.

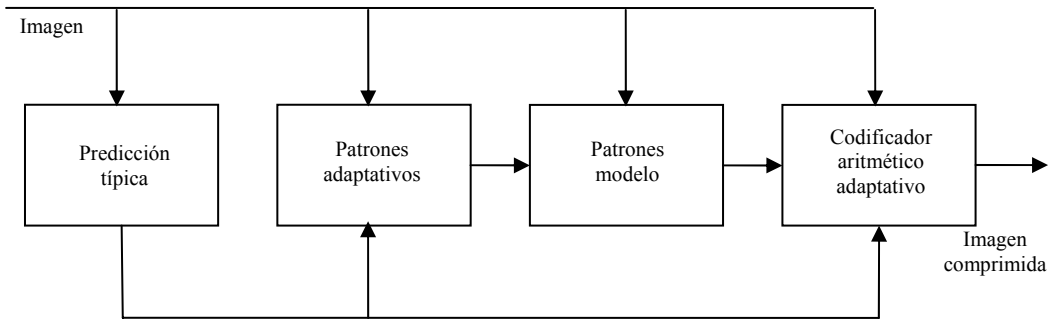


Figura 5-4 : Esquema del algoritmo JBIG

A continuación vamos a describir cada uno de los bloques que componen el compresor JBIG.

5.2.1 Predicción típica

El bloque predicción típica proporciona cierta ganancia de codificación. Este bloque busca regiones de color uniforme, y cuando comprueba que un determinado píxel vigente de alta resolución para codificación se halla en dicha región, no se necesita ninguno de los procesamientos normalmente efectuados en los bloques plantillas adaptativas, plantillas modelo y codificador aritmético adaptativo. En las imágenes de tipo texto o de dibujo lineal, la predicción típica permite usualmente evitar la codificación en el 95% de los píxeles. En las imágenes binivel que reproducen la escala de grises, las economías de procesamiento son considerablemente menores.

5.2.2 Patrones adaptativos

El bloque plantillas adaptativas proporciona una ganancia de codificación sustancial (a veces llega a un factor de 80%) en imágenes que reproducen la escala de grises por semitonos. El patrón adaptativo busca una periodicidad horizontal en la imagen, y cuando la encuentra cambia la plantilla de manera que el píxel que precede al píxel vigente en exactamente con esta periodicidad se incorpore a la plantilla.

Estos cambios son infrecuentes, y cuando se produce uno, se multiplexa una secuencia de control para formar el tren de datos de salida. Por tanto, los decodificadores no necesitan efectuar ningún procesamiento para buscar la formación correcta de los patrones adaptativos.

5.2.3 Patrones modelo

Para cada píxel a codificar, el bloque plantillas modelo proporciona al codificador aritmético un número que indica el contexto. Este número viene determinado por los colores (niveles binarios) de los unos píxeles concretos de la imagen. Los dos modelos de vecindad que se usan son, el *patrón de tres líneas* y el *patrón de dos líneas*.

El codificador aritmético mantiene para cada contexto una estimación de la probabilidad condicional del símbolo dado por ese contexto. La máxima ganancia de codificación se obtiene cuando esta estimación de probabilidad es al mismo tiempo exacta y próxima a 0 ó 1. Así, las buenas plantillas tienen un buen valor predictivo, de manera que cuando los valores de los píxeles de la plantilla son conocidos, el valor del píxel a codificar es altamente predecible.

5.2.4 Codificador aritmético adaptativo

El bloque codificador aritmético adaptativo es un codificador entrópico. Recibe las salidas del bloque predicción típica para determinar incluso si es necesario codificar un determinado píxel. Suponiendo que así es, lee entonces el contexto y utiliza su estimador de probabilidad interno para estimar la probabilidad condicional de que el

píxel vigente sea de un color dado. A menudo el píxel es altamente predecible a partir del contexto, de manera que la probabilidad condicional es muy próxima a 0 ó 1, y puede tenerse una gran ganancia de codificación de entropía.

Mantener las estimaciones de probabilidad para cada uno de los contextos es un problema estadístico nada insignificante. Debe llegarse a un equilibrio entre la obtención de estimaciones sumamente exactas y la necesidad contrapuesta de adaptarse rápidamente a estadísticas subyacentes cambiantes.

Es posible utilizar esta Especificación para la codificación sin pérdidas de imágenes en escala de grises y en color codificando los planos de bits independientemente, como si cada uno fuera en sí mismo una imagen binivel. Una alternativa buena es transformar la representación de los valores digitales de los píxeles de la imagen de codificación binaria a codificación Gray. Una ventaja de esto es que sólo cambia un bit entre dos palabras de código consecutivas, por lo que la mayoría de los bits de los vecinos dentro de cada plano de bit tienen el mismo valor.

5.3 ALGORITMO SLIC

El algoritmo SLIC[5] es un algoritmo de compresión de imágenes en escala de grises sin pérdidas basado en segmentación. La idea de comprimir imágenes segmentando primero, viene fundada en que será más efectivo comprimir regiones de la imagen que sean más homogéneas. Puesto que los bordes de las regiones se pueden representar por matrices unidimensionales, y las regiones serán más o menos homogéneas, se espera que este algoritmo obtenga altas tasas de compresión. La primera complicación que presenta el método es la segmentación. La mayoría de los algoritmos de segmentación son sofisticados y poseen buen comportamiento sólo para un reducido número de imágenes. Para superar el problema de la sofisticación y de la dependencia del tipo de imagen, se adopta para el crecimiento de regiones un algoritmo simple para el crecimiento de regiones. [Singh, 97]

Cuando se hace el proceso de crecimiento de regiones obtenemos una *matriz de discontinuidad, una imagen error* y un vector *que almacenara los bits altos de la semilla*. Por lo tanto pasamos de tener un solo conjunto de datos, a tener 3 conjuntos de datos. La imagen error será la encargada de almacenar la diferencia de un píxel con su píxel central. La matriz de discontinuidad se encargará de aportar información sobre los bordes de las regiones. Los bits altos de la semilla servirán para almacenar los bits que no pueden ser almacenados en la matriz de error, por su tamaño más grande, para ser almacenados en un vector auxiliar.

[5] SLIC : Segmentation-based Lossless Image Compression

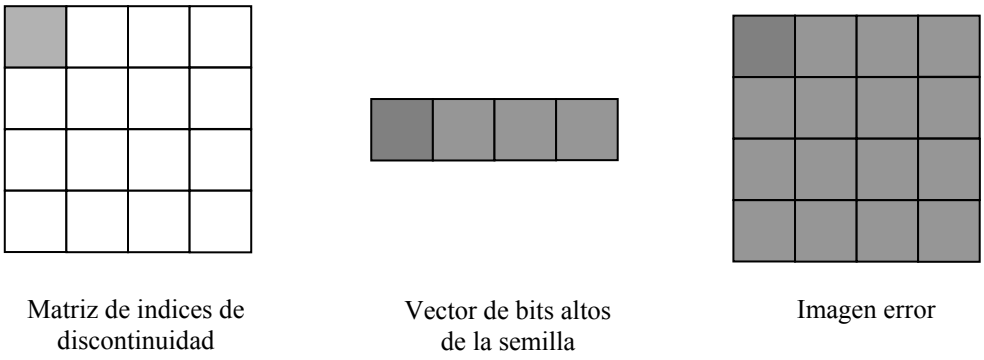


Figura 5-5 : Tres tipos de datos generados por el crecimiento de regiones.

De estos tres tipos de datos que obtenemos tras la segmentación sólo dos de ellos se codificarán, estos son, la matriz de índices de discontinuidades y la matriz de errores. El codificador entrópico que se usa es el codificador JBIG. El vector de bits altos de semilla se incluye directamente en el fichero final.

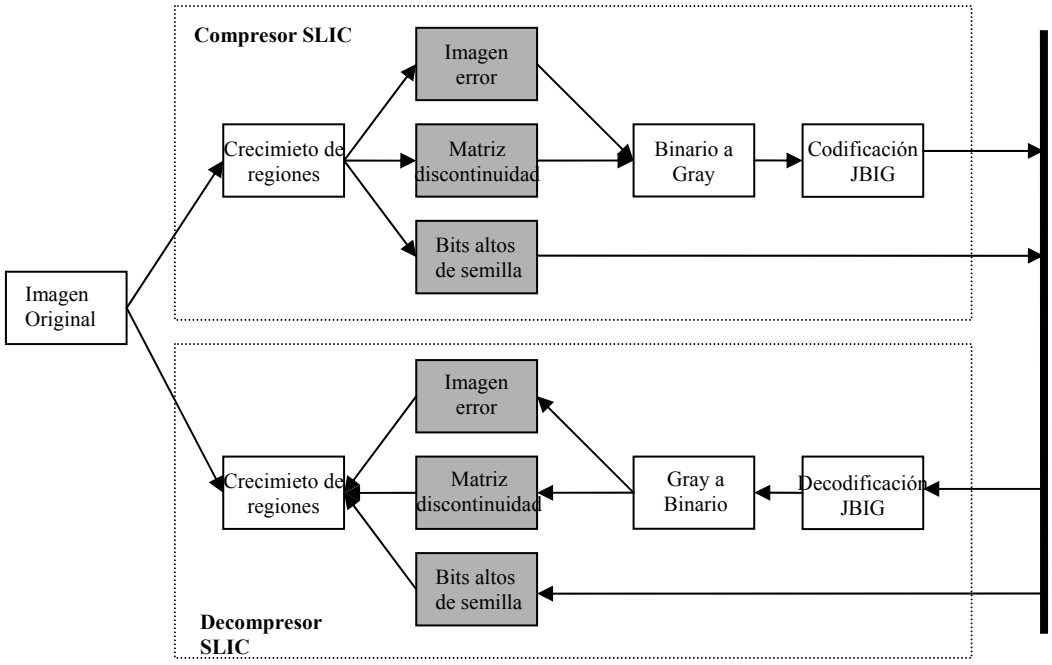


Figura 5-6 : Esquema del compresor SLIC

En el proceso de crecimiento de regiones vecindad 4 se empieza tomando una *semilla* que será el píxel superior izquierdo de la imagen, que en realidad sería el primer píxel de la imagen si consideráramos el sentido de los ejes en una imagen. Este píxel semilla ya forma parte de la primera región. A continuación este píxel se convierte en el píxel central de sus píxeles vecinos, y se comprueba si estos 4 píxeles vecinos pueden o no formar parte de la región.

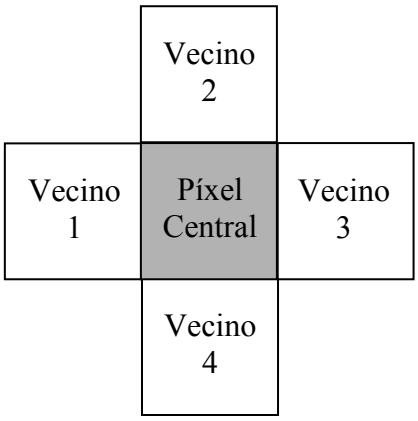


Figura 5-7 : Orden de recorrido de los 4 píxeles vecinos de un píxel central para comprobar la posible inclusión en la región

Si alguno de los vecinos cumplen las condiciones de inclusión en la región, éstos se incluyen en la cola de la cual saldrán en orden de como han entrado, posee una estructura FIFO. Una vez sale un píxel que había sido almacenado en la cola, éste juega el papel de *píxel central*, que a su vez comprobará si sus vecinos pueden o no introducirse en la región, y éstos en caso afirmativo serán incluidos en la cola para que después sean también píxeles centrales. Ni que decir tiene que un píxel que ya ha sido incluido en una región no puede ser incluido en otra, por lo tanto no se comprueba aquellos píxeles que hayan sido incluidos en una región y sean vecino de un píxel central que está comprobando si se puede incluir o no. Cuando la cola de píxeles centrales se vacía, significa que no hay píxeles centrales para seguir creciendo la región, por lo tanto dicha región ya está crecida y hay que empezar a crecer otra. La siguiente región comenzará con el primer píxel que se encuentre rastreando de derecha a izquierda y de arriba abajo que no haya sido incluido en ninguna región aún. El proceso de crecimiento de regiones termina cuando no hay ningún píxel que no pertenezca a alguna región.

Las condiciones de inclusión de un píxel en una región de este algoritmo se resumen en estas dos:

1. Que el píxel vecino el cual estamos estudiando su inclusión en la región no forme parte ya de otra región ya crecida o de la misma.

2. Que el valor absoluto de la diferencia de intensidad de dicho píxel y las correspondiente al píxel central sean menores que un *nivel de error* definido.

La regla de comparación de la segunda condición puede venir especificada por la siguiente ecuacion

$$p\ (i,j) < nivel\_error_l$$

siendo *p(i,j)* el valor de nivel de cada píxel.

El parámetro que se relaciona con el nivel de error es el de *bits de error* cuya relación con el anterior es la siguiente:

$$nivel\,de\,error = 2^{bits\,de\,error} - 1$$

Para almacenar la información del crecimiento necesitaremos dos variables, una que represente el *error* y otra que represente el *índice de discontinuidad*. Empecemos por el índice de discontinuidad, el cual es un escalar, si un píxel va a entrar en una región, y éste entra o no, la información de si ese píxel ha entrado o no se representa por un escalar. Entonces para cada píxel tendremos un valor, esto quiere decir que tendremos una matriz de tamaño el número de píxeles, y que sus componentes serán escalares. En esta matriz se representa el número de veces que un píxel ha intentado entrar en una región y no lo ha conseguido, por lo tanto el valor irá de 0 a 4.

La otra variable es el error. Al ser el error la diferencia de intensidad del píxel en cuestión y la intensidad del píxel central, tendremos que el error es una cantidad escalar. La variable error se convierte en una tabla para cada píxel, que en conjunto estamos hablando de una matriz de tamaño el número de píxeles de la imagen.

Tenemos pues dos matrices, la *matriz de discontinuidades* y la *matriz de error*. A continuación vamos a ver como se están llenando esas matrices. Aquí interviene la variable bits de error, si tenemos un determinado número de bits de error, el valor que tome una posición de la matriz de error, no será mayor que el nivel de error, que a su vez está relacionado con el número de bits de error. Por lo tanto, el máximo número de bits diferentes de 0 estará en función del número de bits de error, en concreto, no podrá superar el número de bits de error. Llegamos a la conclusión que en la matriz de error sólo se almacena el número de bits de error. Si estamos hablando de los píxeles que son semilla de la región, hay que almacenar no sólo la diferencia en la imagen error sino que hay que almacenarlo completo. En el caso de un píxel vecino de uno central, sólo se almacena la diferencia. Para solucionar este problema, los píxeles semilla tendrán un tratamiento especial. Cada píxel semilla lleva asociado una posición en el vector de bits altos de semilla; cada una de estas componentes lleva los bits que no pueden ser almacenados en la matriz de errores, esto es, la diferencia entre 8 y el número de bits de error. El resto se almacena normalmente en la matriz de errores.

CAPÍTULO 6

ALGORITMOS DE COMPRESIÓN DE IMÁGENES SIN PÉRDIDAS DESARROLLADOS

En este apartado vamos a explicar cada uno de los algoritmos de compresión sin pérdidas desarrollados en este proyecto. Se trata de buscar una línea a seguir o descartar otras, a la hora de usar unos u otros métodos de reducción de las distintas redundancias que existen en la imagen, para observar el buen o mal funcionamiento de distintas combinaciones de los métodos de reducción de redundancia.

Vamos a hacer un análisis de los algoritmos desarrollados, los cuales tendrán muchas partes en común. Dichas partes en común se explicarán previamente a la enumeración de cada uno de los esquemas de compresión y su explicación específica.

Todo esquema de compresión está basado en la reducción de las distintas redundancias que posee una imagen, por lo tanto a la imagen se le aplicarán en cada esquema de compresión, algoritmos cuya función sea la de reducir cada una de esas redundancias.

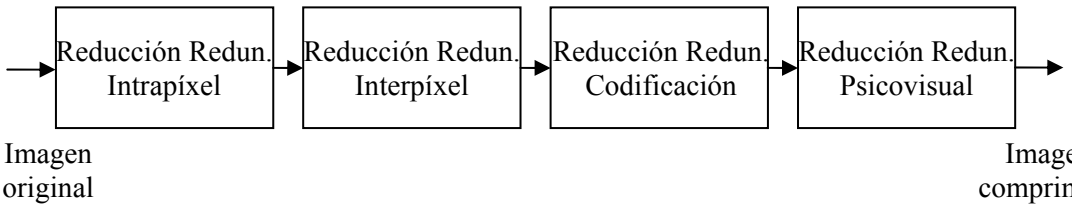


Figura 6-1 : Esquema básico de un esquema de compresión general.

En el esquema de la Figura 6-1, vemos cada uno de los bloques que reducirá la redundancia que le corresponda. Como explicamos en la parte de teoría de la

compresión, si estamos hablando de compresión de imágenes sin pérdidas, esto es, que la imagen recuperada después de la descompresión sea exactamente la misma que la de partida, no deben existir bloques que introduzcan pérdidas. El esquema de la Figura 6-1 es un esquema general de compresión, que es válido tanto para compresión con o sin pérdidas. El elemento, que seguro que sí se encuentra en un esquema de compresión introduciría pérdidas, es el bloque de reducción de redundancia psicovisual. Esta redundancia está basada, entre otras cosas, en que el ojo humano no es capaz de distinguir los 2^{24} colores de una imagen en color real, ni tampoco es capaz de distinguir una reducción de la resolución de la imagen, por esto, para la percepción del sistema visual humano es lo mismo reducir este número de colores. El bloque tendría básicamente un cuantificador, el cual reduciría el amplio número de colores y también la resolución. Por supuesto una reducción o modificación de los niveles de colores así como un cambio en la resolución introducen claramente una pérdida en la compresión. Por todo esto un esquema de compresión sin pérdidas de imágenes tendrá el esquema de la Figura 6-2.

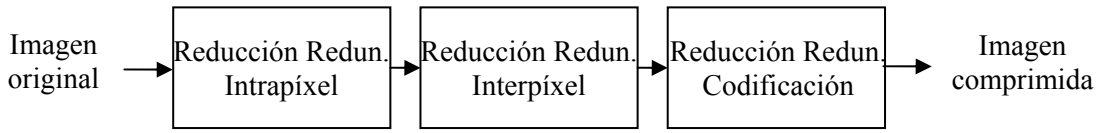


Figura 6-2 : Esquema básico de compresión de imágenes sin pérdidas

Este esquema que tenemos en la Figura 6-2, además de haber eliminado el bloque de eliminación de redundancia psicovisual, que es el que introduciría pérdidas de manera segura, hay que asegurarse que los métodos empleados en los distintos bloques sean de entre todos los existentes, aquéllos que no introducen pérdidas en la imagen. Por ejemplo hay métodos como los ya explicados en el bloque de reducción interpíxel, que supondrían una introducción de pérdidas, a saber, determinadas transformadas de la imagen como la DCT (Transformada discreta del coseno).

En todos los esquemas que se van a presentar a continuación, existen bloques comunes que van a ser comentados previamente a los demás. Estos bloques son dos la transformación al espacio de color *YIQ* y otro, la determinación automática del número de bits de error. Después de describir los bloques comunes se describirán los demás, teniendo éstos ya sus elementos diferenciadores.

6.1 BLOQUES COMUNES DE LOS ESQUEMAS DE COMPRESIÓN

6.1.1 Reducción de la Redundancia Intrapíxel: Sistema de Coordenadas YIQ

Como se dijo en el apartado de compresión, el sistema *RGB* de representación directa de imágenes posee una alta redundancia entre los tres planos de color. Por ello para un sistema de compresión esta alta redundancia que en general existe ha de ser eliminada o por lo menos disminuida. Para ello se aplican transformadas a la imagen; muchas de estas transformaciones a otro espacio de color se utilizan ampliamente en los algoritmos de compresión con pérdidas.

La transformada de imagen que tiene como fin la compresión, ya sea con o sin pérdidas, debe primeramente separar las componentes que llevan la información de color de la imagen, esto es, de la crominancia, de las que llevan información de luminancia. Normalmente las componentes en alta frecuencia de la imagen están concentradas en la componente de luminancia de la imagen. También es muy importante, además de que las componentes de luminancia y crominancia estén desacopladas, que las componentes de crominancia sean ortogonales entre sí.

Uno de los sistemas que cumplen lo anteriormente dicho es el espacio de color *YIQ,* en el cual la componente de luminancia *Y* concentra la información de alta frecuencia de la imagen, esto es, los bordes y contornos. Las componentes *I* y *Q* son las componentes de color que son ortogonales entre sí.

Para hacer un cambio de coordenadas de color de *RGB* a *YIQ* se emplea la transformación lineal [Lim, 90], que viene representada por la siguiente matriz que ya la mostramos cuando explicábamos los espacios de color:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Observando la transformación lineal representada en la matriz que vemos inmediatamente encima de estas líneas, llegamos a la conclusión que, aunque este sistema de color *YIQ* sea muy bueno por las características mencionadas anteriormente para la compresión, lo será para la compresión con pérdidas porque se usa aritmética en punto flotante, cosa que está totalmente prohibida en compresión sin pérdidas porque después habrá que usar un cuantizador para su almacenamiento como valor entero. Pero existe una transformación *YIQ* basada en aritmética entera, y que en consecuencia es completamente reversible. Las ecuaciones que caracterizan dicha transformación son [Singh, 97]:

$$Y_S = \left\lfloor \frac{\left\lfloor \frac{R+G}{2} \right\rfloor + G}{2} \right\rfloor$$

$$I_S = R - G$$

$$Q_S = \left\lfloor \frac{R+G}{2} \right\rfloor - G$$

donde $\lfloor . \rfloor$ denota el operador suelo.

Se han empleado los subíndices *s* a las componentes *YIQ* para diferenciarlas de las normales e indicar que éstas son sin pérdidas. Los autores de esta transformación aritmética demuestran su aproximación calculando el RMSE (Root Mean Square Error) entre las dos imágenes y el error medio relativo, esto es, el error de cada píxel ponderado con el valor de la intensidad de dicho píxel.

Imagen de prueba	*RMSE*	*Error medio relativo (%)*
Imagen 1	8.30	6.0
Imagen 2	8.27	7.0
Imagen 3	4.59	3.0
Imagen 4	3.37	2.0
Imagen 5	4.21	2.5

Tabla 6-1 : Comparación entre las coordenadas YIQ y las coordenadas $Y_sI_sQ_s$ en términos de señal-ruido para 5 imágenes.

Para ver que la transformación es cerrada a continuación mostramos la transformación inversa:

$$R = Y_S + \left\lfloor \frac{Q_S + 1}{2} \right\rfloor + \left\lfloor \frac{I_S + 1}{2} \right\rfloor$$

$$G = Y_S - \left\lfloor \frac{Q_S}{2} \right\rfloor$$

$$B = Y_S + \left\lfloor \frac{Q_S + 1}{2} \right\rfloor - \left\lfloor \frac{I_S}{2} \right\rfloor$$

Como puede observarse en la transformación directa las componentes de crominancia *I* y *Q* pueden tomar valores negativos tras la transformación, con lo que será necesario un bit adicional para representar la información del signo. Para ello se usa una nueva matriz llamada *matriz signo*, en la cual se representa el signo de aquellas componentes susceptibles de tener un valor negativo. Esta matriz estará compuesta del menor valor de representación por el número de píxeles, que debido a su poca información será muy bien comprimida por el codificador.

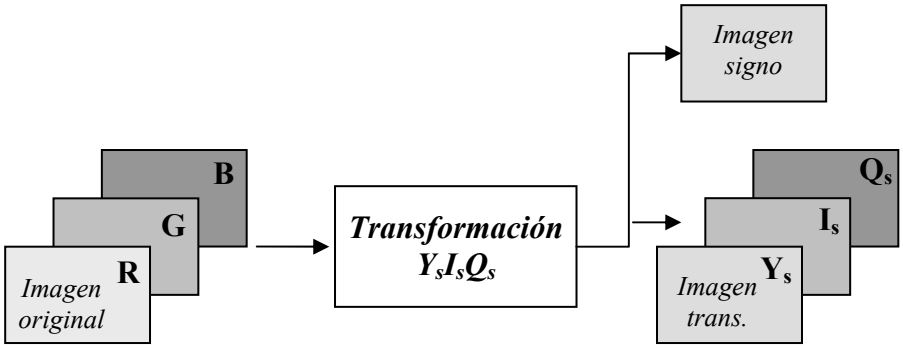


Figura 6-3 : Esquema de transformación de una imagen en el espacio de color RGB al $Y_sI_sQ_s$ junto con la imagen signo.

6.1.2 Algoritmo automático de cálculo de bits de error para cada componente

Para obtener un algoritmo que calculara el mejor número de bits necesarios para cada componente, habría que, suponiendo que los tres valores de bits de error fueran igual para las tres componentes, ejecutar el algoritmo 8 veces y después elegir el que mejor compresión hubiera tenido, pero eso no es factible ya que el tiempo de compresión sería muy alto. Eso suponiendo que los tres valores de bits de error de las tres componentes fueran iguales, lo que en realidad no es así porque en la transformación aritmética $Y_sI_sQ_s$ hace concentrar más información en la componente Y, que necesitará pues más bits de error para ser representado.

El valor de la entropía de cada componente se corresponderá con el número de bits de error, después de haber aplicado el operador techo $\lceil . \rceil$ teniendo en cuenta que el número de bits de error esta comprendido entre 2 y 8. A esta conclusión se llegó después de haber realizado experimentos comprimiendo las imágenes con los diferentes valores de bits de error, y comparando estos resultados con los obtenidos por la entropía [Serrano ,01].

A continuación vamos a presentar los valores de la entropía de varias imágenes que serán aquellas que se usarán para los resultados del proyecto. Vamos a comparar la entropía de la imagen en los espacios de color *RGB* y $Y_sI_sQ_s$:

Imágenes	Y_s	I_s	Q_s
Foto 1	6,9	5,2	4,3
Foto 2	7,1	5,3	3,1
Foto 3	6,1	5,2	4,3
Foto 4	6,5	5,1	4,2
Media	**6,6**	**5,2**	**4,0**
Alemania	7,7	5,6	4,3
Lena	7,4	6,1	4,9
Loros	7,2	6,4	5,9
Pimientos	7,6	6,5	6,2
Media	**7,5**	**6,1**	**5,3**
Mand	6,5	1,2	1,1
Yinywthm	3,8	3,7	3,2
7thm	4,1	3,7	3,8
Atlantis	7,2	6,6	4,9
Media	**5,4**	**3,8**	**3,2**

Tabla 6-2: Tabla de la entropía de cada componente YIQ de diferentes imágenes.

Imágenes	R	G	B
Foto 1	6,8	6,8	7,2
Foto 2	7,5	7,1	6,6
Foto 3	6,4	6,3	6,5
Foto 4	6,7	6,7	6,4
Media	**6,9**	**6,7**	**6,7**
Alemania	7,6	7,7	7,6
Lena	7,3	7,6	7,0
Loros	7,5	7,5	7,2
Pimientos	7,3	7,5	7,1
Media	**7,4**	**7,6**	**7,2**
Mand	6,5	6,4	6,5
Yinywthm	3,8	3,6	4,0
7thm	4,4	4,2	4,9
Atlantis	7,7	7,5	6,2
Media	**5,6**	**5,4**	**5,4**

Tabla 6-3: Tabla de la entropía de cada componente RGB de diferentes imágenes.

El valor que usaremos será el techo de los valores no enteros que hemos obtenido. Destacar también la reducción de redundancia en la imagen y como se ve que en el espacio de color *RGB* todas sus componentes tienen aproximadamente la misma información, pero mucha de ella redundante, y como en el espacio de color $Y_sI_sQ_s$ la componente que posee mayor información es la *Y*.

El método para el cálculo de la entropía es el que presentábamos como cálculo de la entropía como estimación de primer orden. Un esquema del algoritmo para la automatización del cálculo del número de bits de error es el siguiente:

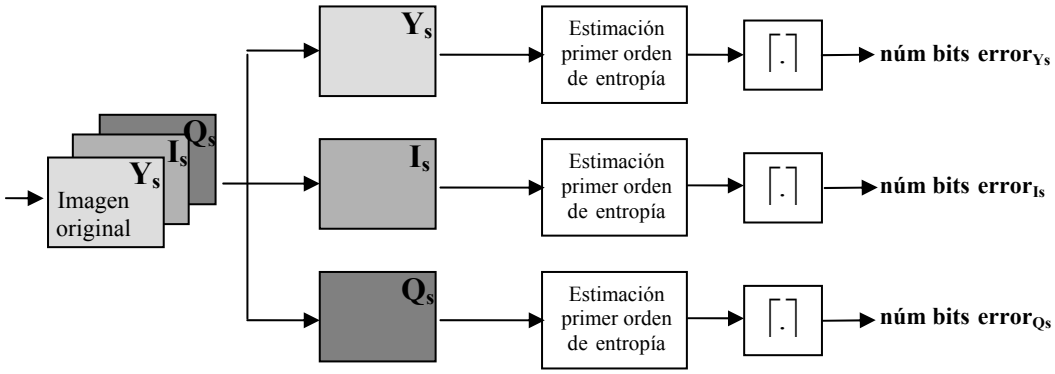


Figura 6-4 : Esquema del cálculo automático de número de bits de error para cada componente basándose en la entropía

6.2 SLCIC. ALGORITMO DE COMPRESIÓN DE IMÁGENES BASADO EN SEGMENTACIÓN CON CRECIMIENTO DE REGIONES DE VECINDAD 4 Y CODIFICADOR JBIG

En este apartado vamos a describir el algoritmo que servirá de base sobre el cual realizaremos las sucesivas transformaciones. Este algoritmo esta basado a su vez en el algoritmo SLIC que se usaba para imágenes en blanco y negro. El SLCIC es la versión ampliada a color del SLIC. Este algoritmo realiza una segmentación de la imagen con el fin de obtener regiones más homogéneas. El esquema general lo vemos en la Figura 6-5.

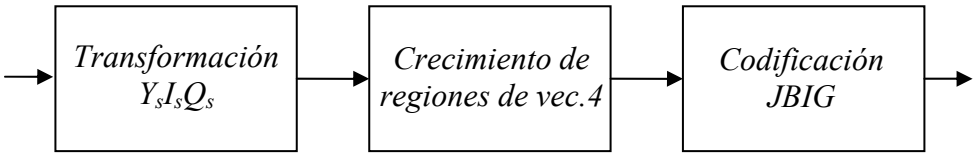


Figura 6-5 : Esquema de compresión con crecimiento de regiones de vecindad 4

Este esquema de compresión SLCIC, posee el algoritmo de determinación automática del número de bits de error de cada componente. Este algoritmo se explicó en los bloques comunes de todos los esquemas y tal como se dijo estará presente en este y en los sucesivos esquemas de compresión. Del mismo modo en este esquema se utiliza la transformación $Y_sI_sQ_s$, lo que reportará un número de bits de error diferentes para cada componente. Esto es debido a cómo la transformación concentra más

información en la componente *Y* que en las otras dos componentes. Por ello la componente *Y* necesitará más números de bits de error para ser representado.

El último bloque es el codificador entrópico JBIG. Este codificador posee además un decorrelador que ayudará mucho a la compresión posterior del codificador en sí.

A continuación se describe el algoritmo de crecimiento de regiones de vecindad 4 que es usado en este esquema de compresión. [Serrano, 01]

6.2.1 Descripción del crecimiento de regiones de vecindad 4

El proceso de crecimiento de regiones es análogo al empleado en el algoritmo SLIC pero extendido a las tres componentes. Se empieza tomando una *semilla* que será el píxel superior izquierdo de la imagen. Este píxel semilla ya forma parte de la primera región. A continuación este píxel se convierte en el píxel central de sus píxeles vecinos, y se comprueba si estos 4 píxeles vecinos pueden o no formar parte de la región .Si alguno de los vecinos cumplen las condiciones de inclusión en la región, éstos se incluyen en la cola de la cual saldrán en el orden en que han entrado, o lo que es lo mismo, que la cola posee una estructura FIFO. Una vez sale un píxel que había sido almacenado en la cola, éste juega el papel de píxel *central*, que a su vez comprobará si sus vecinos pueden o no introducirse en la región, y éstos en caso afirmativo serán de nuevo incluidos en la cola para que después sean también píxeles centrales. Un píxel que ya ha sido incluido en una región no puede ser incluido en otra, por lo tanto no se comprueban aquellos píxeles que hayan sido incluidos en una región y sean vecinos de un píxel central que está comprobando si se puede incluir o no. Cuando la cola de píxeles centrales se vacía, significa que no hay píxeles centrales para seguir creciendo la región, por lo tanto dicha región ya está crecida y hay que empezar a crecer otra. La siguiente región comenzará con el primer píxel que se encuentre rastreando de derecha a izquierda y de arriba abajo que no haya sido incluido en ninguna región aún. El proceso de crecimiento de regiones termina cuando no hay ningún píxel que no pertenezca a alguna región.

Las condiciones de inclusión de un píxel en una región se resumen en estas dos:

1. Que el píxel vecino del que estamos estudiando su inclusión en la región no forme parte ya de otra región ya crecida o de la misma.

2. Que el valor absoluto de la diferencia de intensidades de las tres componentes de dicho píxel y las correspondientes al píxel central sean menores que un *nivel de error* definido para cada una de las componentes.

La regla de comparación de la segunda condición puede venir especificada por las siguientes ecuaciones:

$$p_1(i,j) < nivel\_error_1 \quad AND \quad p_2(i,j) < nivel\_error_2 \quad AND \quad p_3(i,j) < nivel\_error_3$$

siendo $p_k(i,j)$ el valor de cada nivel de cada píxel.

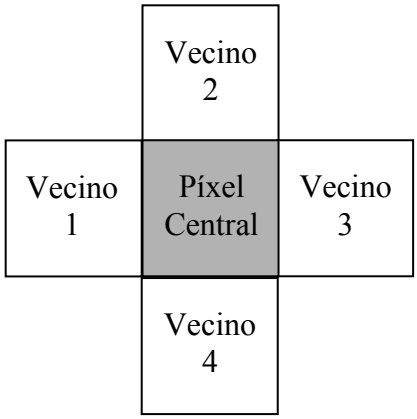


Figura 6-6 : Orden de recorrido de los 4 píxeles vecinos de un píxel central para comprobar la posible inclusión en la región

El nivel de error, responsable de la inclusión de los píxeles en la región viene dado distinto para componente. El parámetro que se relaciona con el nivel de error es el de *bits de error* cuya relación con el anterior es la siguiente:

$$nivel\,de\,error = 2^{bits\,de\,error} - 1$$

el número de bits de error se define para cada plano de color, porque algunas componentes del sistema de representación de la imagen tendrán más o menos información, con lo que la diferencia entre un píxel y otro necesitará, para las componentes que tengan más información en alta frecuencia, más bits para representar la diferencia.

Para almacenar la información del crecimiento necesitaremos dos variables, una que represente el *error* y otra que represente el *índice de discontinuidad*. Empecemos por el índice de discontinuidad, el cual es un escalar debido a que el crecimiento de regiones es vectorial, es decir, se crece la imagen con sus tres planos de color a la vez, lo que implica que si un píxel va a entrar, y éste entra o no, la información de si ese píxel ha entrado o no se representa por un escalar. Entonces para cada píxel tendremos un valor, esto quiere decir que tendremos una matriz de tamaño el número de píxeles, y que sus componentes serán escalares. En esta matriz se representa el número de veces que un píxel ha intentado entrar en una región y no lo ha conseguido, por lo tanto el valor irá de 0 a 4.

La otra variable es el error. Al ser el error la diferencia de cada una de las componentes del píxel en cuestión y las componentes del píxel central, puesto que son tres las componentes de las que se compone cada píxel, tendremos que el error es una cantidad vectorial de tamaño 3. La variable error se convierte en una tabla para cada píxel, que en conjunto estamos hablando de una matriz de tamaño el número de píxeles de la imagen y cada componente de la matriz tiene 3 elementos.

Tenemos pues dos matrices, la *matriz de discontinuidades* y la *matriz de error*. A continuación vamos a ver como se están llenando esas matrices. Aquí interviene la variable bits de error, si tenemos para una componente un determinado número de bits de error, el valor que tome una posición de la matriz de error en una de sus componentes, no será mayor que el nivel de error, que a su vez está relacionado con el número de bits de erroor la ecuación anteriormente expuesta:

$$nivel\,de\,error = 2^{bits\,de\,error} - 1$$

por lo tanto, el máximo número de bits diferentes de 0 en cada elemento estará en función del número de bits de error, en concreto, no podrá superar el número de bits de error. Llegamos a la conclusión de que en la matriz de error sólo se almacena en cada componente el número de bits de error que viene determinado para esa componente. No necesitaría más explicación si estamos hablando de los píxeles que no son semilla de la región, ya que en el caso de que el píxel sea el semilla no se almacena en la imagen error como los demás, sino que se almacena parte del píxel en la imagen error y la parte restante en el vector de bits altos de semilla, hay que almacenarlo entero porque no tiene relación con ningún otro. Para solucionar este problema, los píxeles semilla tendrán un tratamiento especial. Cada píxel semilla lleva asociado una posición en un vector de dimensión 3; cada una de estas componentes lleva para cada componente los bits que no pueden ser almacenados en la matriz de errores, esto es, para cada componente la diferencia de 8 y el número de bits de error. El resto se almacena normalmente en la matriz de errores.

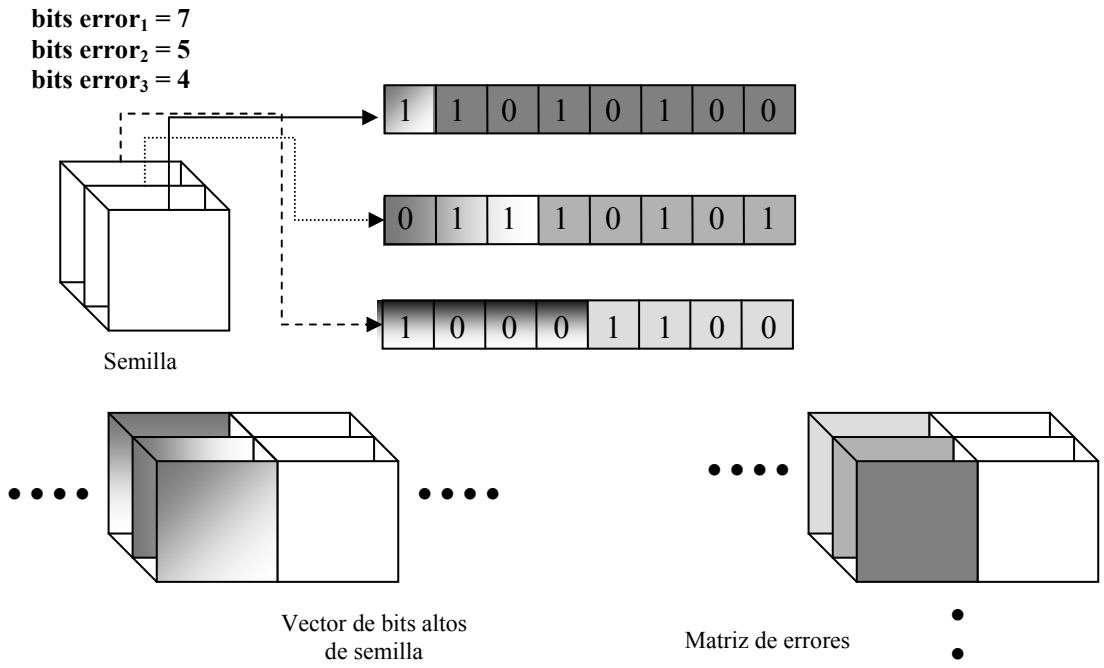


Figura 6-7 : Esquema de almacenamiento de la información de los bits semilla

Como hemos presentado en la Figura 6-7 los píxeles que son semillas de las regiones tienen un tratamiento especial, que queda resuelto con un *vector de bits altos de semilla*.

En un crecimiento de regiones de una imagen tenemos como resultado tres variables, que son : *imagen error*, *matriz de índices de discontinuidad* y *vector de bits altos de semilla*.

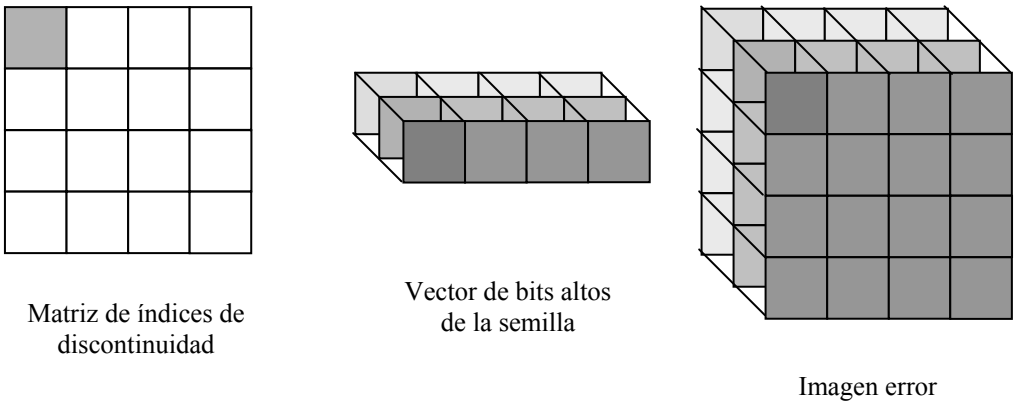


Figura 6-8 : Resultado de la segmentación de la imagen con la matriz de discontinuidad, el vector de bits altos de la semilla y la imagen error

En el algoritmo de decodificación se debe recuperar la imagen a partir de estas tres variables. Se sigue un crecimiento de regiones con los tres conjuntos de datos que tenemos. Se comienza por el píxel superior izquierdo el cual será el primer píxel central; el píxel será clave para la reconstrucción, ya que el valor del píxel vecino depende del píxel central que creció a dicho píxel vecino. El valor del píxel vecino será valor píxel central más error del píxel en cuestión. El proceso de crecimiento de regiones es similar al de codificación y se puede resumir en los siguientes pasos:

1. Se selecciona el píxel semilla rastreando de izquierda a derecha y de arriba abajo, aquel píxel que no haya sido incluido en ninguna región.

2. Una vez seleccionado un píxel semilla, éste se convierte en píxel central y comprueba si sus 4 vecinos cumplen la condición para ser incluidos en la región y pasar a la cola para después ser él mismo un píxel central. Puede suceder que sea incluido en la región o no, todo ello dependerá de la condición de inclusión. Esta condición se compone de dos condiciones:

    i. El píxel vecino que está en estudio de ser incluido no debe haber sido crecido en ninguna otra región previamente.

    ii. El índice de discontinuidad del píxel que se va a incluir debe ser igual a 0.

3. Mirar en la cola si hay píxeles, si los hay se saca el siguiente que corresponda y se convierte en píxel central y seguiríamos en el paso 2. Si por el contrario no queda ningún píxel restante se pasa al paso 1 ya que la región ya ha terminado de crecerse y habrá que buscar otro píxel semilla.

El valor del píxel que estamos creciendo es la suma del valor del píxel central más el valor de la posición de dicho píxel en la imagen error para la correspondiente componente.

A continuación vamos a ver un ejemplo gráfico de crecimiento de regiones para entender mejor su funcionamiento. Partimos de una imagen

46 114 151	45 115 151	55 110 151	84 176 184	55 111 153
45 115 151	45 115 151	55 110 151	52 112 152	55 111 153
45 115 154	45 115 154	51 110 151	48 110 154	55 110 151
48 114 152	48 114 152	50 110 151	52 115 153	54 113 152
55 114 151	55 114 151	49 116 152	48 111 152	55 112 152

Figura 6-9 : Parte de una imagen que se utilizará posteriormente para el estudio de resultados

46 114 151 *(semilla)*	45 115 151 -1 1 0	55 110 151 *(semilla)*	84 176 184 *(semilla)*	55 111 153 0 0 0
45 115 151 -1 1 0	45 115 151 0 0 0	55 110 151 0 0 0	52 112 152 -3 2 1	55 111 153 3 -1 1
45 115 154 0 0 3	45 115 154 0 0 3	51 110 151 *(semilla)*	48 110 154 -3 0 3	55 110 151 0 -1 -2
48 114 152 3 -1 -2	48 114 152 3 -1 -2	50 110 151 -1 0 0	52 115 153 -2 1 1	54 113 152 -1 3 1
55 114 151 *(semilla)*	55 114 151 0 0 0	49 116 152 *(semilla)*	48 111 152 *(semilla)*	55 112 152 1 -1 0

(a)

6 2 7	3 5 4	7 6 7	4 0 0	4 4 4
3 5 4	4 4 4	4 4 4	1 6 5	7 3 5
4 4 7	4 4 7	3 6 7	1 4 7	4 3 2
7 3 2	7 3 2	3 4 4	2 6 5	3 7 5
7 2 7	4 4 4	1 4 0	0 7 0	5 3 4

(b)

0	0	1	3	0
0	0	1	0	0
0	0	2	3	0
0	0	2	0	0
1	1	2	3	0

(c)

5 14 17	6 13 17	10 22 23	6 13 35	6 14 17	6 14 17	6 14 35	6 13 35

(d)

Figura 6-10 : Crecimiento de regiones de una imagen 5X5 con vecindad 4. El parámetro bits de error se ha fijado a 3 para las tres componentes. (a) Resultado del crecimiento. (b) Matriz de índices de discontinuidad. (c) Imagen Error. (d) Vector de los bits altos de las semillas.

Cada casilla del resultado del crecimiento contiene 2 triadas de números, una corresponde al valor del píxel y la otra corresponde al error, resultado de haber restado cada una de las componentes del píxel a su correspondiente píxel central. Hacer notar que hay valores negativos, para evitar esto lo que se hace es sumarle a todos los errores la mitad del rango de *nivel de error*, esto es, si hemos usado para el ejemplo un número de bits de error de 3 para las tres componentes, el nivel de error será 8, que repartiéndolo entre positivo y negativo hace que el intervalo de diferencia haya sido [-4,3] , por lo que para que el valor se encuentre en el intervalo de [0,7] es necesario sumarle 4 a todos los errores.

6.2.2 Esquema general de compresión: Transformación YsIsQs. Crecimiento de regiones vecindad 4. Codificación JBIG.

Los bloques de los que está compuesto el esquema de compresión SLCIC son la transformación del espacio de color, el crecimiento de regiones de vecindad 4 y la codificación JBIG.

En el primer bloque lo que se persigue es la reducción de la redundancia intrapíxel existente entre los tres planos de color *RGB*. Esta transformación de espacio de color concentra una mayor cantidad de información en la componente *Y* y menor en las otras dos componentes. Este hecho puede beneficiar al crecimiento de regiones dándole un número mayor de bits de error a la componente de luminancia, o lo que es lo mismo, un menor número de bits de error a las componentes de crominancia. Pero el valor de este número lo determinará la estimación de primer orden de la entropía que realicemos previamente al crecimiento.

En el segundo bloque del esquema tenemos el crecimiento de regiones de vecindad 4. Este bloque reduce la redundancia interpíxel. A este bloque se le pasa la imagen transformada y después de crecer la imagen, da como resultado la imagen error, la matriz de discontinuidad y el vector de bits altos de semilla. Al codificador JBIG se le pasa la imagen error, la matriz de discontinuidad y la matriz signo, esta última es resultante de la transformación de espacio de color. El vector de bits altos de semilla no se codifica y se incluye directamente en el fichero de salida.

El último bloque del esquema de compresión es el codificador JBIG. A este bloque se le deben pasar para que sea más efectivo los datos en código Gray. Por lo tanto previamente a codificarlo, debemos pasar un bloque para transformar el código binario a código Gray. Ésta transformación sólo debe aplicarse a la imagen error, la matriz de discontinuidades y la matriz signo. El vector de bits altos de semilla ni se codifica ni se transforma a código Gray.

En la Figura 6-11 y Figura 6-12 se muestra un esquema más detallado de la compresión.

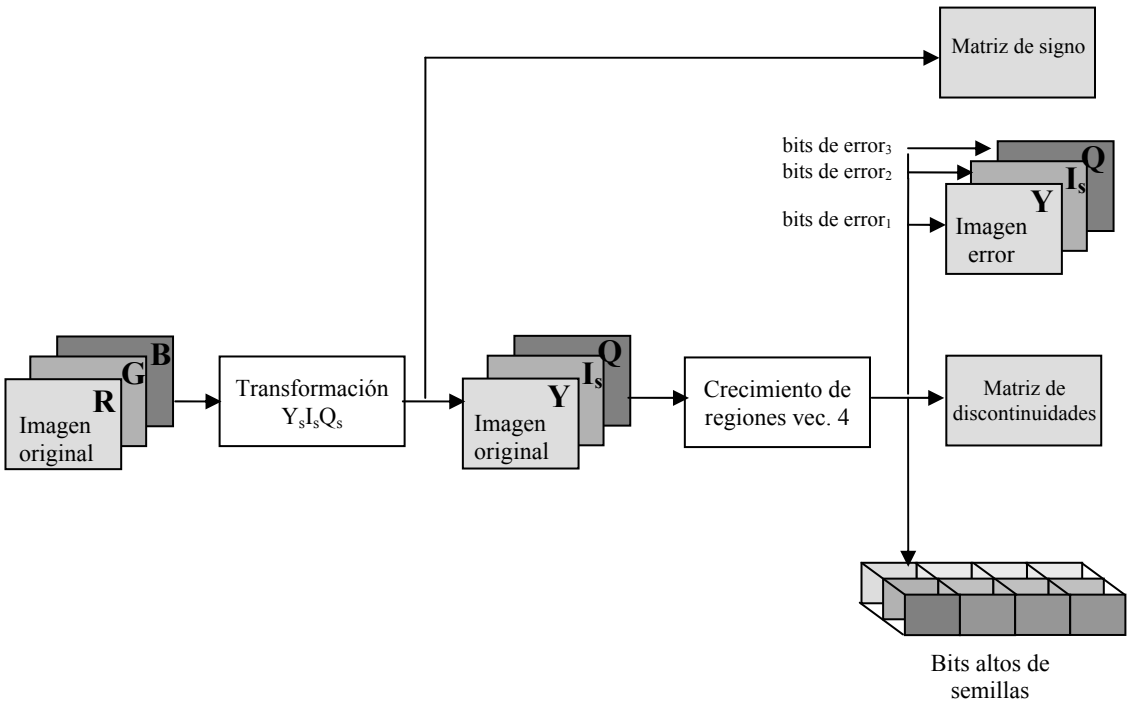


Figura 6-12: Ilustración detallada del esquema de compresión hasta el crecimiento de regiones

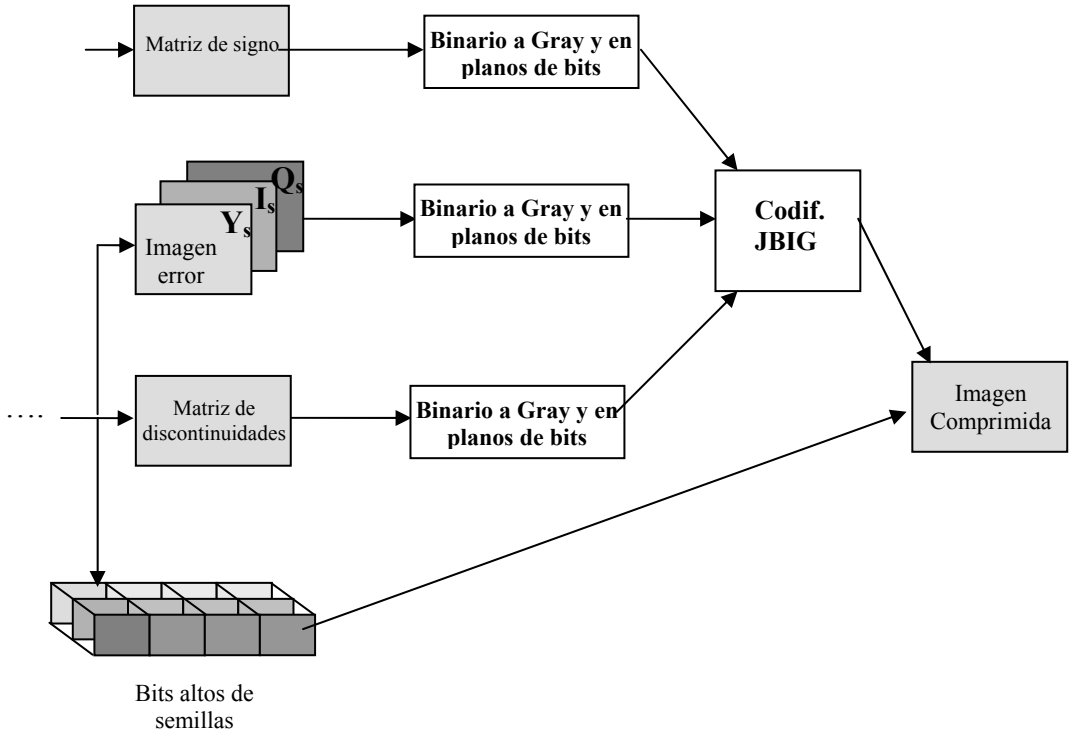


Figura 6-11: Ilustración detallada del esquema de compresión en la zona de codificación

En el extremo en el que se decodifique la imagen se transforma de Gray a binario,

se recompone el crecimiento de regiones y cuando tenemos la imagen en el espacio YIQ lo transformamos con la transformación inversa de aritmética entera.

6.2.3 Resumen

En este apartado se ha explicado el algoritmo SLCIC, basado en un transformación del espacio de color, un crecimiento de regiones de vecindad 4 y un codificador JBIG.

6.3 ALGORITMO DE COMPRESION DE IMÁGENES BASADO EN SEGMENTACION CON CRECIMIENTO DE REGIONES DE VECINDAD 8

Este esquema de compresión esta basado en SLCIC[6] [Serrano, 01] que a su vez está basado en el algoritmo SLIC[7], compresor de imágenes basado en segmentación de imágenes en niveles de gris. Este algoritmo realiza un proceso de segmentación a la imagen en regiones para obtener regiones más homogéneas, y en cada una de las regiones tenemos un píxel semilla a partir del cual vamos almacenando en él la diferencia entre un píxel y su central.

Podemos estructurar en bloque este esquema de compresión como muestra la Figura 6-13.

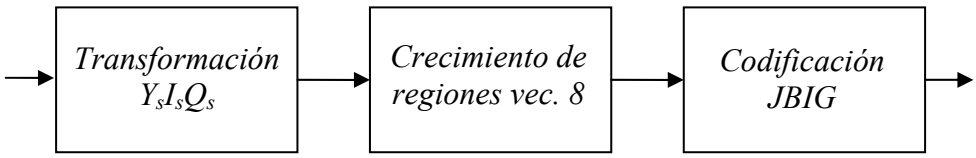


Figura 6-13 : Esquema de compresión con crecimiento de regiones de vecindad 8

6.3.1 Descripción del crecimiento de regiones de vecindad 8

El proceso de crecimiento de regiones es análogo al empleado en el algoritmo SLIC así como en el algoritmo SLCIC, se parece más a este último en el que se usa vecindad 4. Tanto en uno como en otro se empieza tomando una *semilla* que será el píxel superior izquierdo de la imagen, que en realidad sería el primer píxel de la imagen si consideráramos el sentido de los ejes en una imagen. Este píxel semilla ya forma parte de la primera región. A continuación este píxel se convierte en el píxel central de sus píxeles vecinos, y se comprueba si estos 8 píxeles vecinos pueden o no formar parte de la región .Si alguno de los vecinos cumplen las condiciones de inclusión en la región,

[6] Siglas de Segmentation-based Lossless Color Image Compression. Algoritmo de compresión de imágenes a color basada en segmentación.

[7] Segmentation-based Lossless Image Compression. Algoritmo de compresión sin pérdidas basado en segmentación para imágenes en niveles de gris.

éstos se incluyen en la cola de la cual saldrán en el orden en que han entrado, posee una estructura FIFO. Una vez sale un píxel que había sido almacenado en la cola, éste juega el papel de píxel *central*, que a su vez comprobará si sus vecinos pueden o no introducirse en la región, y éstos en caso afirmativo serán de nuevo incluidos en la cola para que después sean también píxeles centrales. Un píxel que ya ha sido incluido en una región no puede ser incluido en otra, por lo tanto no se comprueban aquellos píxeles que hayan sido incluidos ya en una región y sean vecinos de un píxel central que está comprobando si se puede incluir o no. Cuando la cola de píxeles centrales se vacía, significa que no hay píxeles centrales para seguir creciendo la región, por lo tanto dicha región ya está crecida y hay que empezar a crecer otra. La siguiente región comenzará con el primer píxel que se encuentre rastreando de derecha a izquierda y de arriba abajo que no haya sido incluido en ninguna región aún. El proceso de crecimiento de regiones termina cuando no hay ningún píxel que no pertenezca a alguna región.

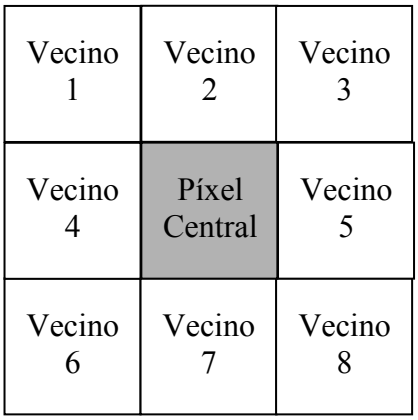


Figura 6-14 : Orden de recorrido de los 8 píxeles vecinos de un píxel central para comprobar la posible inclusión en la región

Las condiciones de inclusión de un píxel en una región de este algoritmo se resumen en estas dos:

1. Que el píxel vecino del que estamos estudiando su inclusión en la región no forme parte ya de otra región ya crecida o de la misma.

2. Que el valor absoluto de la diferencia de intensidades de las tres componentes de dicho píxel y las correspondientes al píxel central sean menores que un *nivel de error* definido para cada una de las componentes.

La regla de comparación de la segunda condición puede venir especificada por las siguientes ecuaciones:

$$p_1(i,j) < nivel\_error_1 \quad AND \quad p_2(i,j) < nivel\_error_2 \quad AND \quad p_3(i,j) < nivel\_error_3$$

siendo $p_k(i,j)$ el valor de cada nivel de cada píxel.

El nivel de error, responsable de la inclusión de los píxeles en la región viene dado distinto para componente. El parámetro que se relaciona con el nivel de error es el de *bits de error* cuya relación con el anterior es la siguiente:

$$nivel\,de\,error = 2^{bits\,de\,error} - 1$$

el número de bits de error se define para cada plano de color, porque algunas componentes del sistema de representación de la imagen tendrán más o menos información, con lo que la diferencia entre un píxel y otro necesitará, para las componentes que tengan más información en alta frecuencia, más bits para representar la diferencia.

Para almacenar la información del crecimiento necesitaremos dos variables, una que represente el *error* y otra que represente el *índice de discontinuidad*. Empecemos por el índice de discontinuidad, el cual es un escalar debido a que el crecimiento de regiones es vectorial, es decir, se crece la imagen con sus tres planos de color a la vez, lo que implica que si un píxel va a entrar, y éste entra o no, la información de si ese píxel ha entrado o no se representa por un escalar. Entonces para cada píxel tendremos un valor, esto quiere decir que tendremos una matriz de tamaño el número de píxeles, y que sus componentes serán escalares. En esta matriz se representa el número de veces que un píxel ha intentado entrar en una región y no lo ha conseguido, por lo tanto el valor irá de 0 a 8.

La otra variable es el error. Al ser el error la diferencia de cada una de las componentes del píxel en cuestión y las componentes del píxel central, puesto que son tres las componentes de las que se compone cada píxel, tendremos que el error es una cantidad vectorial de tamaño 3. La variable error se convierte en una tabla para cada píxel, que en conjunto estamos hablando de una matriz de tamaño el número de píxeles de la imagen y cada componente de la matriz tiene 3 elementos.

Tenemos pues dos matrices, la *matriz de discontinuidades* y la *matriz de error*. A continuación vamos a ver como se están llenando esas matrices. Aquí interviene la variable bits de error, si tenemos para una componente un determinado número de bits de error, el valor que tome una posición de la matriz de error en una de sus componentes, no será mayor que el nivel de error, que a su vez está relacionado con el número de bits de error. por la ecuación anteriormente expuesta:

$$nivel\,de\,error = 2^{bits\,de\,error} - 1$$

por lo tanto, el máximo número de bits diferentes de 0 en cada elemento estará en función del número de bits de error, en concreto, no podrá superar el número de bits de error. Llegamos a la conclusión de que en la matriz de error sólo se almacena en cada componente el número de bits de error que viene determinado para esa componente. Tenemos un tratamiento diferente para los píxeles semilla, ya que en el caso de que el píxel sea semilla no se almacena por la diferencia entre dos píxeles que sí que se podría almacenar con el número de bits de error, estamos hablando de que hay que almacenarlo entero porque no tiene relación con ningún otro anterior al ser el semilla. Para solucionar este problema, los píxeles semilla tendrán un tratamiento especial. Cada píxel semilla lleva asociado una posición en un vector de dimensión 3; cada una de estas componentes lleva para cada componente los bits que no pueden ser almacenados en la matriz de errores, esto es, para cada componente la diferencia de 8 y el número de bits de error, y el resto se almacena normalmente en la matriz de errores.

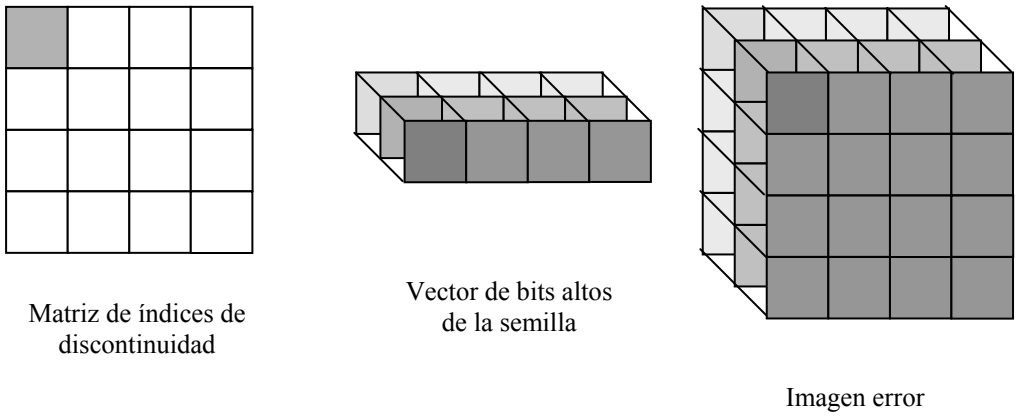


Figura 6-15 : Resultado de la segmentación de la imagen con la matriz de discontinuidad, el vector de bits altos de la semilla y la imagen error

En un crecimiento de regiones de una imagen tenemos como resultado tres variables, que son : *imagen error*, *matriz de índices de discontinuidad* y *vector de bits altos de semilla*.

En el algoritmo de decodificación se debe recuperar la imagen a partir de estas tres variables. Se sigue un crecimiento de regiones con los tres conjuntos de datos que tenemos. Se comienza por el píxel superior izquierdo el cual será el primer píxel central; el píxel será el clave para la reconstrucción, ya que el valor del píxel vecino depende del píxel central que creció a dicho píxel vecino. El valor del píxel vecino será valor píxel central + error del píxel en cuestión. El proceso de crecimiento de regiones es similar al de codificación y se puede resumir en los siguientes pasos:

1. Se selecciona el píxel semilla rastreando de izquierda a derecha y de arriba abajo, aquel píxel que no haya sido incluido en ninguna región.

2. Una vez seleccionado un píxel semilla, éste se convierte en píxel central y comprueba si sus 8 vecinos cumplen la condición para ser incluidos en la región y pasar a la cola para después ser él mismo un píxel central. Puede suceder que sea incluido en la región o no, todo ello dependerá de la condición de inclusión. Esta condición se compone de dos condiciones:

    i. El píxel vecino que está en estudio de ser incluido no debe haber sido crecido en ninguna otra región previamente.

    ii. El índice de discontinuidad del píxel que se va a incluir debe ser igual a 0.

3. Mirar en la cola si hay píxeles, si los hay se saca el siguiente que corresponda y se convierte en píxel central y seguiríamos en el paso 2. Si por el contrario no queda ningún píxel restante se pasa al paso 1 ya que la región ya ha terminado de crecerse y habrá que buscar otro píxel semilla.

El valor del píxel que estamos creciendo es la suma del valor del píxel central más el valor de la posición de dicho píxel en la imagen error para la correspondiente componente.

A continuación vamos a ver un ejemplo gráfico de crecimiento de regiones para entender mejor su funcionamiento. Partimos de una imagen

46 114 151	45 115 151	55 110 151	84 176 184	55 111 153
45 115 151	45 115 151	55 110 151	52 112 152	55 111 153
45 115 154	45 115 154	51 110 151	48 110 154	55 110 151
48 114 152	48 114 152	50 110 151	52 115 153	54 113 152
55 114 151	55 114 151	49 116 152	48 111 152	55 112 152

Figura 6-16 : Parte de una imagen que se utilizará posteriormente para el estudio de resultados.

que corresponde a una parte de una imagen que se utilizará posteriormente para el estudio de resultados.

46 114 151 *(semilla)*	45 115 151 -1 1 0	55 110 151 *(semilla)*	84 176 184 *(semilla)*	55 111 153 0 0 0
45 115 151 -1 1 0	45 115 151 -1 1 0	55 110 151 0 0 0	52 112 152 -3 2 1	55 111 153 3 -1 1
45 115 154 0 0 3	45 115 154 0 0 3	51 110 151 3 -4 -1	48 110 154 -2 0 3	55 110 151 3 -2 -1
48 114 152 3 -1 -2	48 114 152 3 -1 -2	50 110 151 2 -4 -1	52 115 153 3 -1 1	54 113 152 -1 3 1
55 114 151 *(semilla)*	55 114 151 0 0 0	49 116 152 1 2 0	48 111 152 -2 1 1	55 112 152 1 -1 0

(a)

6 2 7	3 5 4	7 6 7	4 0 0	4 4 4
3 5 4	3 5 4	4 4 4	1 6 5	7 3 5
4 4 7	4 4 7	7 0 3	2 4 7	7 2 3
7 3 2	7 3 2	6 0 3	7 3 5	3 7 5
7 2 7	4 4 4	5 6 4	2 5 5	5 3 4

(b)

0	0	2	4	4
0	0	5	1	1
0	0	3	0	2
0	0	0	1	3
2	4	0	0	1

(c)

5 14 17	6 13 17	10 22 23	6 13 35

(d)

Figura 6-17 : Crecimiento de regiones de una imagen 5X5 con vecindad 8. El parámetro bits de error se ha fijado a 3 para las tres componentes. (a) Resultado del crecimiento. (b) Matriz de índices de discontinuidad. (c) Imagen Error. (d) Vector de los bits altos de las semillas.

Cada casilla del resultado del crecimiento contiene 2 triadas de números, una corresponde al valor del píxel y la otra corresponde al error, resultado de haber restado cada una de las componentes del píxel a su correspondiente píxel central. Hacer notar que hay valores negativos, para evitar esto lo que se hace es sumarle a todos los errores la mitad del rango de *nivel de error*, esto es, si hemos usado para el ejemplo un número de bits de error de 3 para las tres componentes, el nivel de error será 8, que repartiéndolo entre positivo y negativo hace que el intervalo de diferencia haya sido [-4,3] , por lo que para que el valor se encuentre en el intervalo de [0,7] es necesario sumarle 4 a todos los errores.

6.3.2 Esquema general de compresión: Transformación $Y_sI_sQ_s$. Crecimiento de regiones vecindad 8. Codificación JBIG.

Después de comentar el crecimiento de regiones de vecindad 8, vamos a ver en que parte del algoritmo introduciremos este bloque, el de segmentación. Como explicábamos anteriormente todo sistema de compresión de imágenes lleva asociado normalmente un sistema de cambio de las componentes de color. El sistema de representación que vamos a usar, como se dijo, en todos los esquemas será el espacio de color $Y_sI_sQ_s$, que como se explicó, era una transformación aritmética con el fin de no introducir pérdidas en la compresión. A continuación nos preguntamos dónde debiera colocarse el bloque de transformación de color; estudios realizados [Serrano, 01] aconsejan que el bloque de transformación de las coordenadas de color debe de colocarse previo al crecimiento de regiones. En estos trabajos así se demostró con un crecimiento de vecindad 4, aquí lo haremos para uno de vecindad 8. Por lo tanto el esquema que vamos a seguir a continuación para explicar este primer esquema de compresión será el que ya expusimos anteriormente y que ahora repetimos:

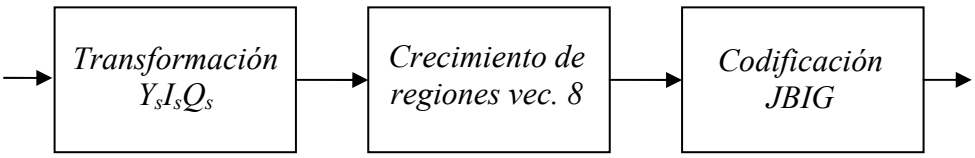


Figura 6-18 : Esquema de compresión de imágenes sin pérdidas

En primer lugar se realiza una transformación al espacio de color $Y_sI_sQ_s$, con el cual se conseguirá reducir la redundancia intrapíxel. La componente de luminancia tendrá más información que las de crominancia, ya que en la primera se almacena la información de alta frecuencia. Este hecho lo podemos utilizar para dar mayor número de bits de error, ya que va a tener más saltos de valores la luminancia, a la componente Y y el que corresponda a las componentes de crominancia. Está función se automatizará

en el siguiente esquema de compresión que se expondrá y que se calculará con la entropía de cada una de las componentes.

En segundo lugar tenemos el crecimiento de regiones. A este se le pasa la imagen transformada, la cual tendrá tres componentes como tenía en el espacio de color *RGB*. A esta imagen transformada se le aplicará el algoritmo de crecimiento de regiones que dará como resultado una imagen error, una matriz de discontinuidades y el vector de bits altos de las semillas, las cuales se aplicarán al codificador JBIG. La transformación al espacio de color *YIQ* genera a su vez una matriz llamada matriz signo como ya apuntábamos, cuya finalidad es la de convertir a la imagen transformada en una tal en la que todos sus componentes sean positivos. La matriz signo no será aplicada al crecimiento de regiones. A continuación presentamos un esquema más detallado de estos dos primeros bloques del esquema.

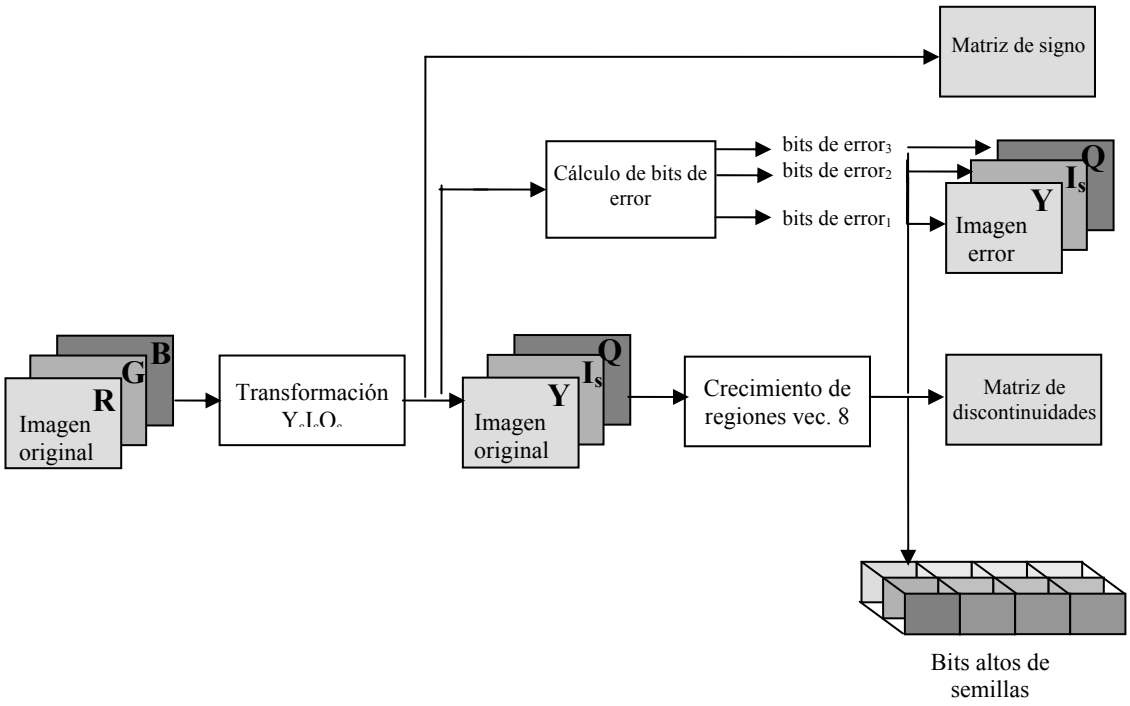


Figura 6-19 : Ilustración detallada del esquema de compresión hasta el crecimiento de regiones

En tercer lugar aplicamos el bloque de reducción de la redundancia en la codificación. El codificador que empleamos es el codificador JBIG. A continuación, la imagen error queda descompuesta en tres matrices escalares que, junto con las demás que ya eran escalares, serán convertidas a código Gray. La razón de esta transformación es que el algoritmo JBIG es más eficiente usando el código Gray. Estas matrices escalares ya en código Gray se dividen en planos de bits que será lo que se le pase al codificador JBIG. De esta forma se consigue una decorrelación de los datos y una codificación entrópica necesaria en todo algoritmo de compresión. Las matrices que son codificadas por el codificador JBIG son la *imagen error*, la *matriz de discontinuidades* y la *matriz de signo*. El vector de bits altos de semilla no es comprimido por el codificador.

En el extremo en el que se decodifique la imagen, los planos de bits se componen de nuevo, se pasa de Gray a binario, se recompone el crecimiento de regiones lo que nos da la imagen en el espacio YIQ. Por último se transforma al espacio de color RGB ayudados por la matriz signo, con lo que se recupera la imagen original.

En la Figura 6-20 se muestra la continuación del esquema de compresión que comprende el cambio a código Gray y su posterior codificación.

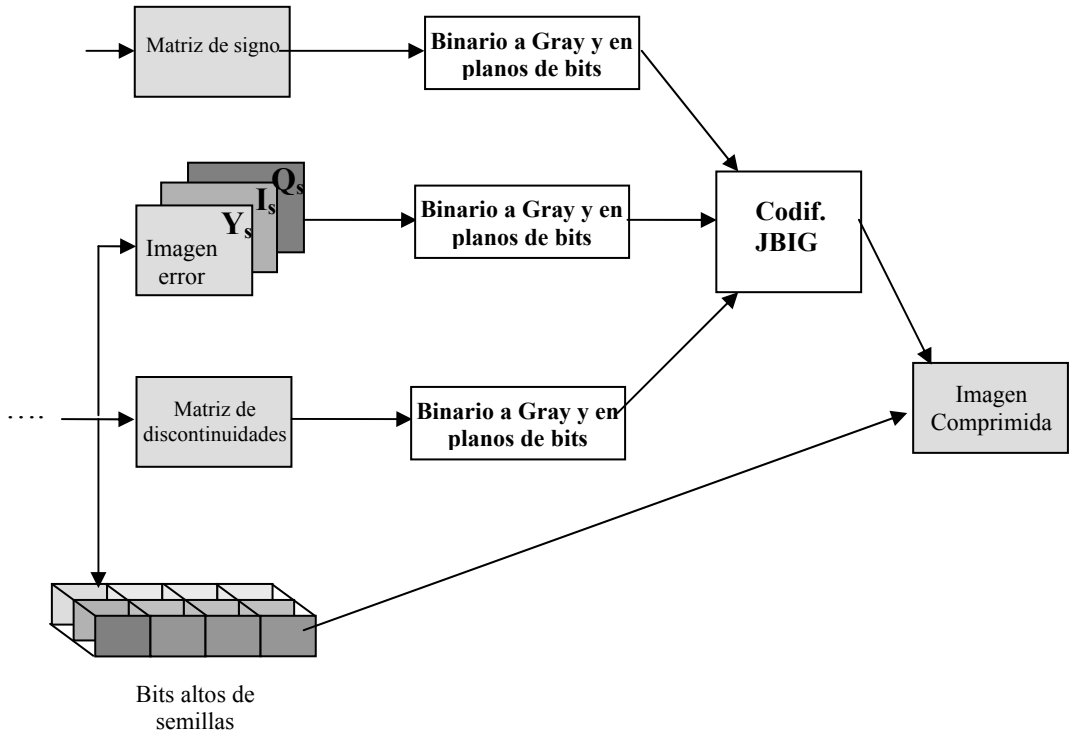


Figura 6-20 : Ilustración detallada del esquema de compresión en la zona de codificación

6.3.3 Resumen

En este apartado hemos desarrollado un algoritmo para comparar el crecimiento de regiones que usa vecindad 4 [Serrano, 01] y el crecimiento con vecindad 8. Todos los demás elementos de los dos esquemas son los mismos. Se buscaba con ello buscar una línea que seguir para posteriores desarrollos, si usar vecindad 4 u 8. A partir de ahora se seguirá usando vecindad 4 ya que como se expondrá en los resultados del proyecto se obtienen mejores tasas de compresión usando un algoritmo de segmentación que use vecindad 4.

6.4 ALGORITMO DE COMPRESIÓN DE IMÁGENES BASADO EN SEGMENTACIÓN CON CRECIMIENTO DE REGIONES DE VECINDAD 4 Y CODIFICACIÓN ARITMÈTICA

En este esquema de compresión vamos a seguir usando la transformación $Y_sI_sQ_s$, así como el crecimiento de regiones, esto es, una segmentación básica, pero en este caso vamos a usar vecindad 4 ya que se demostró que era superior en compresión a la vecindad 8. En la Figura 6-21 presentamos el esquema de compresión.

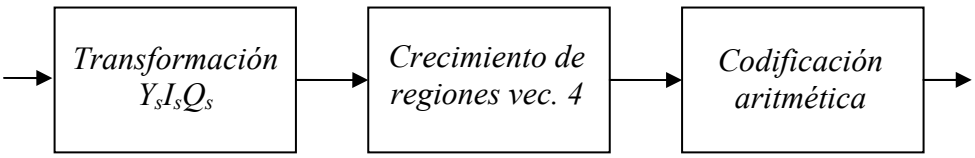


Figura 6-21 : Esquema de compresión basado en transformación $Y_sI_sQ_s$, crecimiento de regiones con vecindad 4 y codificación aritmética.

Como podemos observar además de la modificación en la segmentación, que ahora es de vecindad 4, el codificador lo hemos cambiado por uno aritmético. Este paso de incluir un codificador aritmético en sustitución de un codificador JBIG, es un paso intermedio, porque después habrá que mejorar los bloques previos al codificador aritmético, ya que el codificador JBIG tenía además un decorrelador que habrá que suplir con alguna mejora.

La transformación $Y_sI_sQ_s$ de aritmética entera sigue teniendo su importancia a la hora de actuar como un reductor de la redundancia intrapíxel. De igual modo que en el esquema anterior esta transformación generará una imagen llamada *imagen* $Y_sI_sQ_s$ además de la *matriz signo* que es necesaria porque las componentes de crominancia necesitan un bit más para su representación.

La segmentación que se realiza a la imagen posee el mismo esquema que el realizado en el anterior apartado, con la diferencia que la vecindad es 4, es decir, es el mismo que en el apartado SLCIC.

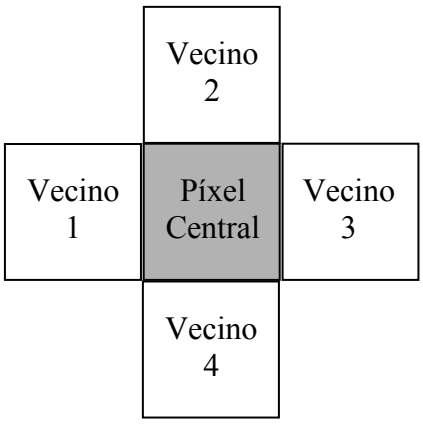


Figura 6-22 : Orden de recorrido de los 8 píxeles vecinos de un píxel central para comprobar la posible inclusión en la región

Si algunos de estos 4 píxeles candidatos para entrar en la región cumplen las dos condiciones: que no formen parte ya de una región, y que el valor absoluto de la diferencia de intensidades de las tres componentes sea menor que el *nivel de error*, entrarán a formar parte de la cola y cuando les toque el turno se convertirán en píxeles centrales.

El *nivel de error* viene determinado por la estimación de primer orden de la entropía de la imagen.

Para almacenar la información del crecimiento tenemos nuevamente la *imagen error* y la *matriz de discontinuidades*, así como el *vector de bits altos de las semillas*. La diferencia fundamental con el algoritmo anterior de vecindad 8 es que el valor máximo que va tomar cualquier elemento de la matriz de discontinuidades va a ser 4, es decir, el intervalo será [0,4].

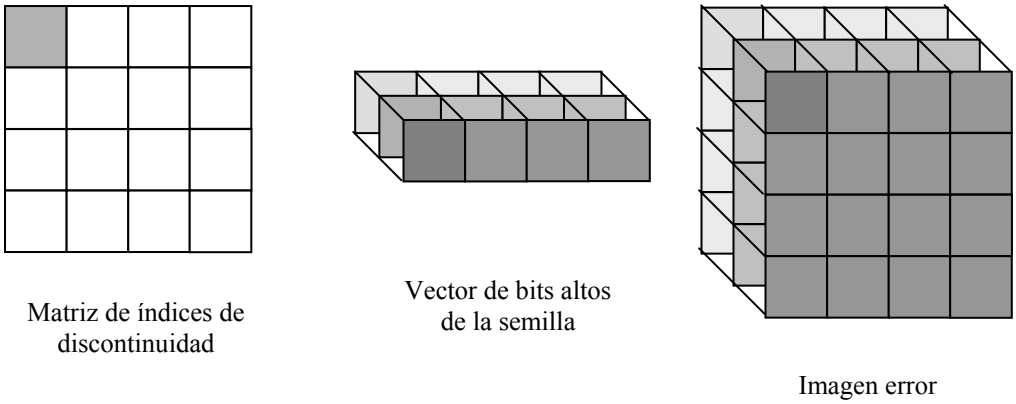


Figura 6-23 : Resultado de la segmentación de la imagen con la matriz de discontinuidad, el vector de bits altos de la semilla y la imagen error

El crecimiento de regiones de vecindad 4 es el mismo que el que se usa en SLCIC por lo tanto no vamos a realizar ningún ejemplo igual que realizamos en el crecimiento de vecindad 4 y de vecindad 8.

6.4.1 Codificador aritmético

En este caso el bloque que vamos a utilizar para reducir la redundancia en la codificación es un codificador aritmético. El codificador de Huffman junto con el codificador aritmético son los dos codificadores entrópicos más conocidos. La ventaja que tiene el codificador aritmético con respecto al de Huffman es que en este último se usa codificación de bloque, cosa que reduce la eficiencia y que en el codificador aritmético no se usa. [Witten, 87]

Al igual que hacíamos con el codificador JBIG vamos a hacer pasar por este bloque a la *imagen error*, la *matriz de discontinuidades* y la matriz que resultaba de la transformación $Y_sI_sQ_s$ que llamamos *matriz de signo*. El vector de los bits altos de las semillas no se codifica como en la versión anterior.

El esquema detallado de la transformación $Y_sI_sQ_s$, el crecimiento de regiones y el codificador aritmético se muestran en la Figura 6-24.

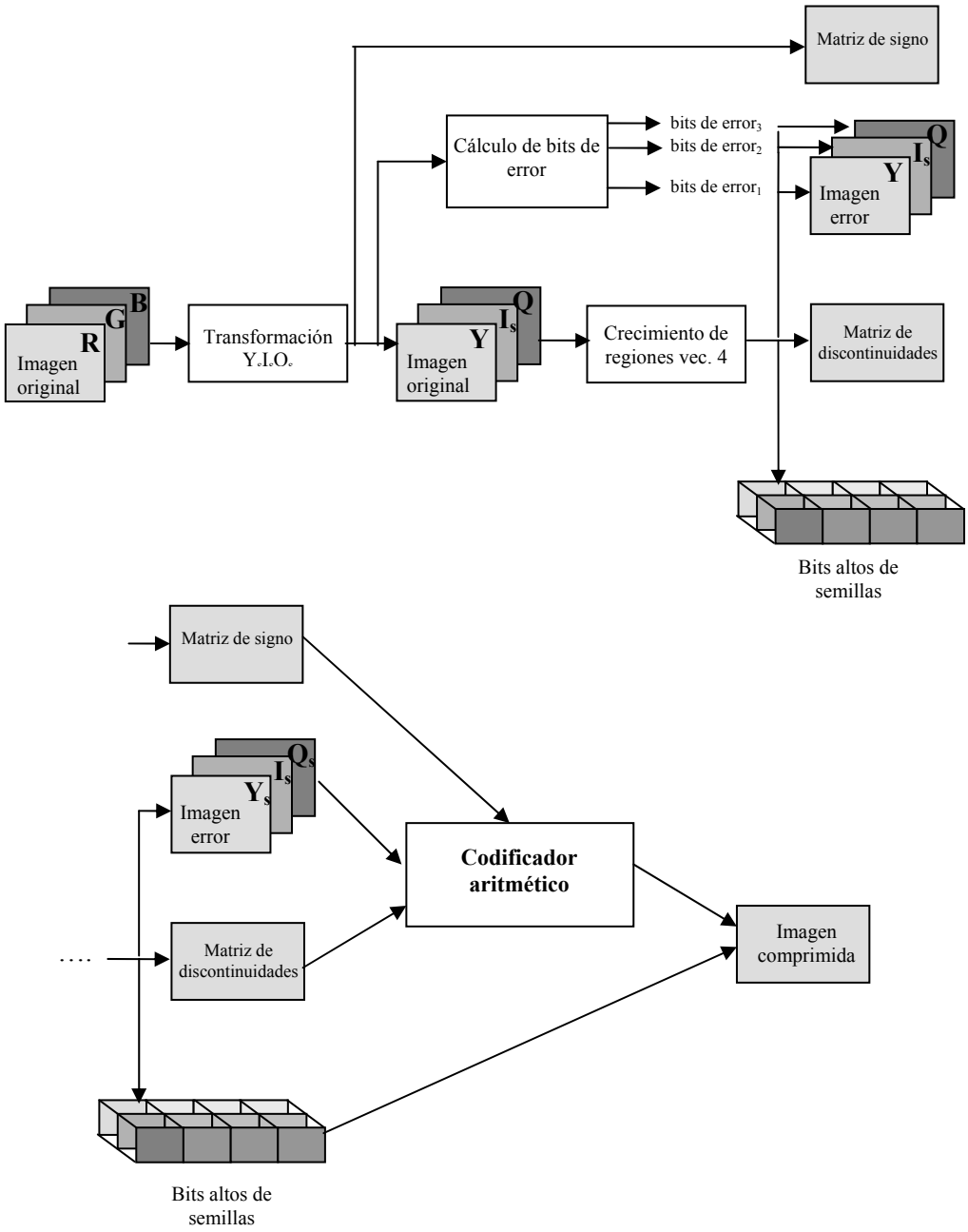


Figura 6-24 : Esquema detallado del proceso de compresión basado en transformación $Y_sI_sQ_s$, calculo automático de los bits de error, crecimiento de regiones con vecindad 4 y codificación aritmética.

6.4.2 Resumen

Después de haber realizado la comparación entre el crecimiento con vecindad 8 y con vecindad 4, hemos llegado a la conclusión de que para el crecimiento de regiones es preferible usar vecindad 4. Hemos introducido un nuevo elemento sustituyéndolo por el codificador JBIG, el codificador aritmético, que a partir de ahora será el que usemos. Esto es debido a que, además de que su eficiencia es muy buena, a continuación vamos

a modificar el algoritmo de crecimiento de forma que el número de bits de error sea variable para intentar reducir aún más la redundancia ya que el codificador aritmético, a pesar de sus buenas características, debería llevar algún bloque que iguale al que poseía el JBIG, esto es, un decorrelador previo a la codificación.

6.5 ALGORITMO DE COMPRESIÓN DE IMÁGENES BASADO EN SEGMENTACIÓN CON CRECIMIENTO DE REGIONES MODIFICADO Y CODIFICACIÓN ARITMÉTICA

En este nuevo esquema de compresión vamos principalmente a desarrollar un algoritmo de crecimiento de regiones que será de vecindad 4, pero el valor de los bits de error variará según el entorno del píxel en cuestión; si vemos que el entorno es más o menos homogéneo reduciremos el numero de bits de error, por el contrario si vemos que no es homogéneo aumentaremos los bits de error. Aumentan todos los valores de bits de error y teniendo cuidado de que no sobrepasen los límites de 2 y 8.Una de las razones por las que se dejó de utilizar el codificador JBIG es porque el número de bits de error debía de ser constante. Aunque el codificador aritmético no es bueno por deméritos del otro, éste también tiene muy buenas características.

En la Figura 6-25 vamos a ver como hemos hecho hasta ahora el esquema de compresión y a ir describiendo los distintos bloques. Algunos de ellos no habrá que comentarlos mucho porque ya se han utilizado anteriormente y el comportamiento en este esquema es el mismo.

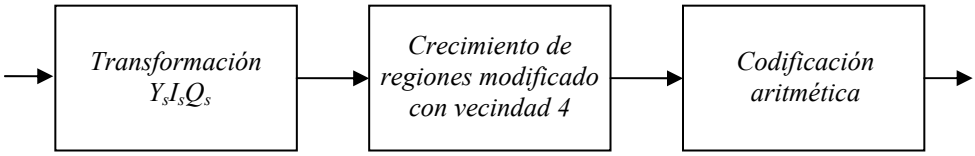


Figura 6-25 : Esquema de compresión basado en transformación $Y_sI_sQ_s$, crecimiento de regiones con vecindad 4 y codificación aritmética.

El primer bloque con el que nos encontramos es la transformación en el espacio de color $Y_sI_sQ_s$ para reducir la redundancia intrapíxel de la imagen, a continuación tenemos en crecimiento de regiones modificado, el cual explicaremos detalladamente a continuación, y como reductor de la redundancia de codificación tenemos el codificador aritmético.

6.5.1 Descripción del crecimiento de regiones modificado de vecindad 4

Lo que se busca con modificar el crecimiento de regiones es que, el número de bits de error que está íntimamente relacionado con el nivel de error que se permite para que un píxel entre en una región, pueda variar dependiendo de la homogeneidad o la diferencia de la imagen en una zona. Si una imagen es homogénea en una zona, el nivel de error puede ser más bajo por lo tanto el número de bits de error en esa zona bajará. En cambio si estamos hablando de una zona de la imagen de mucha actividad, el nivel

de error entre un píxel y otro puede ser más grande de lo normal, y como consecuencia subimos el número de bits de error para representarlo.

Como en el algoritmo de crecimiento normal, cada región se comienza por el primer píxel que no esté dentro de ninguna región, rastreando de izquierda a derecha y de arriba abajo, al cual se le llamara *píxel semilla*. Para el funcionamiento del algoritmo lleva asociado a cada píxel de la imagen una tabla tridimensional, que llamaremos *tabla de crecimiento*, y tantos elementos como píxeles haya, la cual se encargará de almacenar el valor del número de bits de error del píxel central por el que ha sido crecido. Esta tabla no formará parte de la información comprimida, es una tabla auxiliar. Para el píxel semilla se almacena en la tabla el valor de la entropía para cada una de las componentes de la imagen y se almacenará en la imagen error y en el vector de bits altos de semillas. A continuación sabiendo cual es el número de bits de error con el que tiene que crecer un píxel, podemos aplicar el algoritmo normal de crecimiento a los 4 vecinos del píxel central. Las dos reglas para que un píxel sea incluido en una región son:

1. Que el píxel vecino que estamos estudiando su inclusión en la región, no forme parte de otra región ya crecida.

2. Que el valor absoluto de la diferencia de intensidades de las tres componentes de dicho píxel y las correspondientes al píxel central sean menores que un *nivel de error* definido para cada una de las tres componentes.

En este algoritmo el valor del *nivel de error* va variando según estemos por una zona homogénea de la imagen o no. Un píxel central puede crecer una región por sus 4 vecinos, pero dependiendo de la zona en la que se encuentre tendrá un nivel de error mayor o menor, y esto supondrá que los píxeles vecinos, de no pertenecer ya a una región, sean o no incluidos en la región. El valor con el que un píxel central tiene que crecer a sus vecinos, depende de con el número de bits de error con el que fue crecido y con el valor de la matriz de discontinuidades. La matriz de discontinuidades da una idea de cómo de homogénea es una zona de la imagen, si el índice de discontinuidad es 0, significa que ese píxel ha sido crecido en el primer intento por lo que el *nivel de error* que se ha usado podría ser alto. Lo mismo que si hablamos de un valor de índice de discontinuidad 3 por ejemplo, eso significa que ese píxel ha intentado ser incluido en una región 3 veces, por lo que el *nivel de error* se puede llegar a entender que debería ser más alto. Después de haber estudiado experimentalmente el valor óptimo para que el *nivel de error* deba aumentar o disminuir, dependiendo si nos encontramos en una zona homogénea o no de una zona de la imagen es:

- Si el índice de discontinuidad de un píxel central es <=1 antes de intentar incluir en la región a sus 4 vecinos, ha de restar 1 a cada uno de los valores de bits de error de cada una de sus componentes, que son a su vez las del píxel central que creció a este píxel central actual. Pero si alguno de los valores de bits de error de alguna de las componentes es 2, como el límite inferior es 2, no se resta a ninguno de los bits de error de ninguna de las componentes, ya que si no fuera así la descodificación sería imposible.

- Si el índice de discontinuidad de un píxel central es >=3 antes de intentar incluir en la región a sus 4 vecinos, ha de sumar 1 a cada uno de los valores de

bits de error de cada una de sus componentes, que son a su vez las del píxel central que creció a este píxel central actual. Pero si alguno de los valores de bits de error de alguna de las componentes es 8, como el límite superior es 8, no se suma a ninguno de los bits de error de ninguna de las componentes, ya que si no fuera así la descodificación sería imposible.

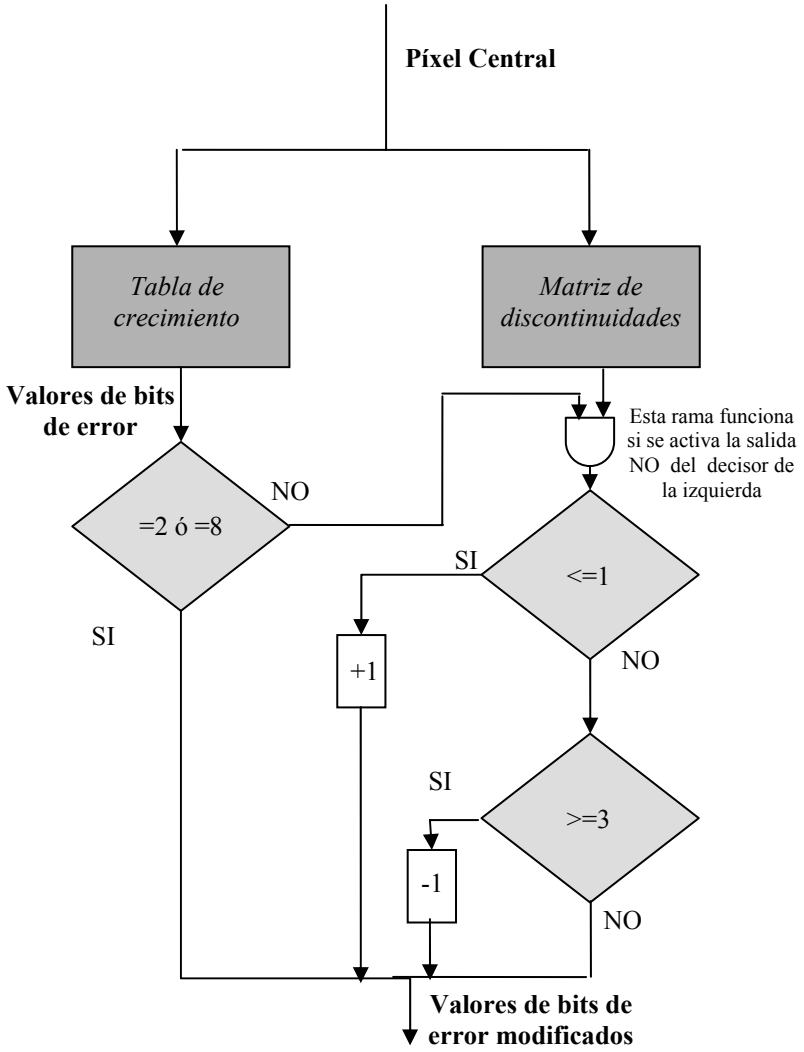


Figura 6-26 : Esquema de modificación del número de bits de error dependiendo de la matriz de discontinuidades.

Una vez que el píxel central ya sabe con el número de bits de error que tiene que continuar creciendo, lo hace como siempre se ha hecho, con el añadido de que, si un píxel es incluido en la región hay que introducir en la *tabla de crecimiento* en la posición del píxel que ha entrado y el valor del número de bits de error del píxel central que le ha crecido. Pero como se dijo esta *tabla de crecimiento* no se incluirá en el archivo que llevará la imagen comprimida, es sólo una tabla auxiliar.

Los conjuntos de datos que dan como resultado el crecimiento modificado, son la *imagen error*, la matriz de *índices de discontinuidad* y el vector de bits altos de semilla. Repetir que los píxeles semilla usan el valor de la entropía para almacenarse en la imagen error y en el vector de bits altos como ya se explicó gráficamente.

Los datos que serán codificados por el codificador aritmético dando lugar a la imagen comprimida. En la Figura 6-27 exponemos un esquema detallado del algoritmo de compresión.

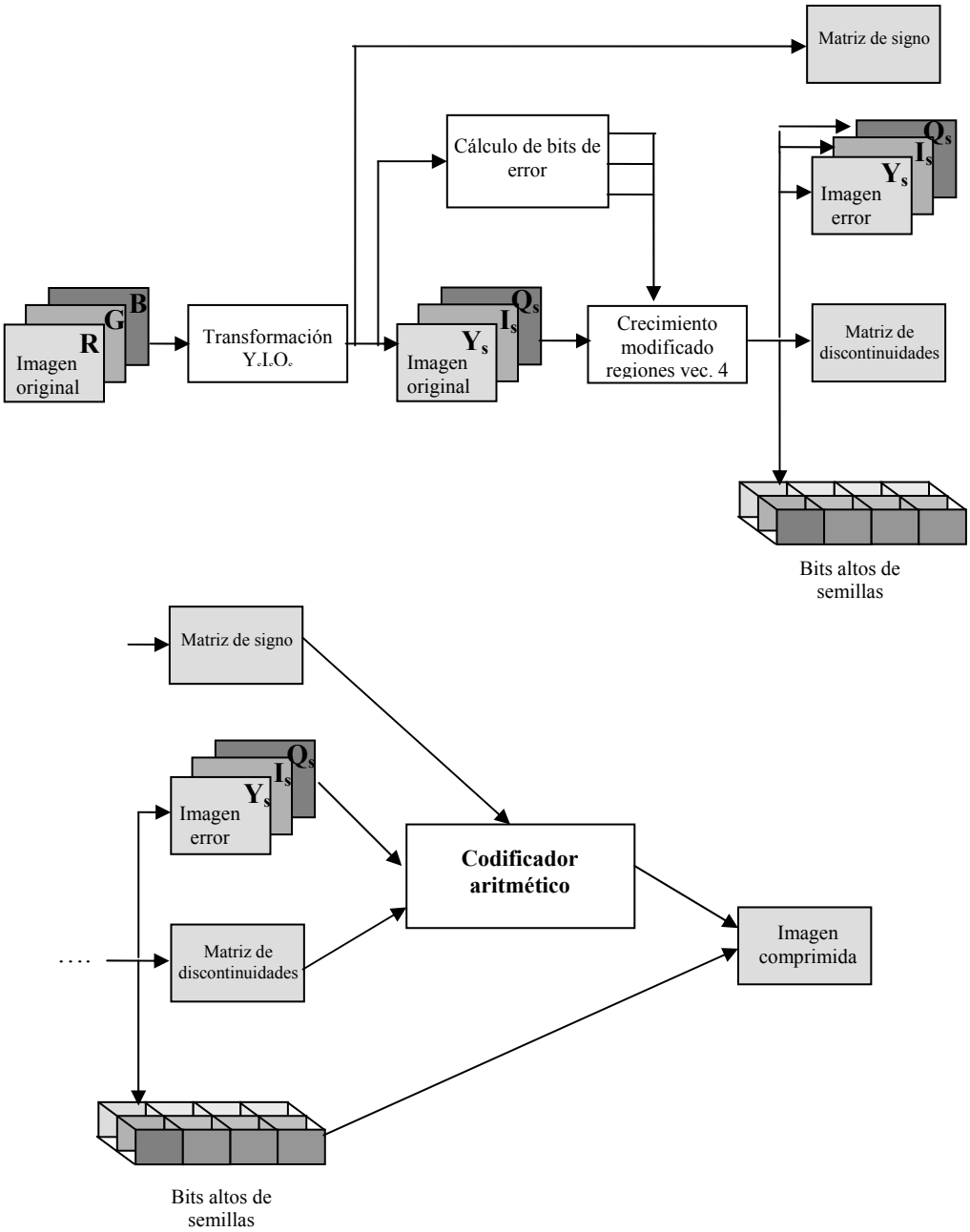


Figura 6-27 : Esquema detallado de el proceso de compresión basado en transformación $Y_sI_sQ_s$, cálculo automático de los bits de error, crecimiento de regiones modificado con vecindad 4 y codificación aritmética.

En el algoritmo de descodificación se lleva a cabo el proceso inverso. El primer paso es descodificar aritméticamente los tres grupos de datos, la matriz signo, la imagen error y la matriz de discontinuidades. El vector de bits altos de semilla no necesita ser descodificado. A continuación se crece de nuevo la imagen con los datos que poseemos. En el crecimiento hay que tener en cuenta que el número de bits de error es variable

dependiendo de la zona de la imagen. Como se ha comentado en otras ocasiones un píxel se generará a partir de otro cuando su índice de discontinuidad sea 0, será entonces cuando se le sumará el valor del píxel que crece al valor del error. Pero tenemos que tener en cuenta que el valor de bits de error tiene mucho que ver a la hora de crecer. Además de sumarle el error que está almacenado en la matriz de error hay que restarle el valor que se necesitaba sumar en el crecimiento para que los valores fueran positivos. Este valor está íntimamente relacionado con el número de bits de error para cada componente. Este valor para cada píxel central dependerá de la zona de la imagen donde se encuentre. Por ejemplo el número de bits de error con que se empieza a crecer una región es el mismo que el valor de la entropía para cada una de las tres componentes, en cambio si este píxel crece a otro, cuando el crecido salga de la cola para convertirse en píxel central puede que tenga otro número de bits de error distintos con los que crecer. Esto se soluciona usando una tabla auxiliar donde se van almacenando los valores de lo bits de error para cada píxel central. Al crecer un píxel central a un vecino, ha de mirar cuál es su índice de discontinuidad del vecino, ya que si éste es menor o igual que 1 se decrementa el número de bits de error en 1 - si ninguno de ellos es 2 ó 8 -, y si es mayor o igual a 3 se incrementa el valor de los bits de error en 1 - también teniendo en cuenta que ninguno de ellos se ni 2 ni 8- para cuando éste sea píxel central. Pero para hacer esto de una forma ordenada hay que tener en cuenta que cuando se va intentando incluir un píxel en una región, se van disminuyendo los índices de discontinuidad hasta llegar a 0 que es cuando se incluye el píxel. Pues bien, necesitaremos un flag para indicar la primera vez que se accede al valor de un índice de discontinuidad y en este momento almacenar el valor de los índices de discontinuidad para cuando haya que sumar o restar 1, o dejarlo igual. La próxima vez que se acceda a ese píxel si el flag está activado significa que los valores de sus índices de discontinuidad ya han sido almacenados. Los posibles valores de éste flag son tres:

- 1 :No se incrementa ni decrementa los valores de bits de error.
- 2: Se incrementa los valores de bits de error en 1.
- 3: Se decrementa los valores de bits de error en 1

Todo esto teniendo en cuenta que los valores de bits de error no puedes sobrepasar 8 por arriba y 2 por abajo. A continuación una vez recuperada la imagen original se transforma al espacio de color RGB.

6.5.2 Resumen

En este capítulo se ha introducido un crecimiento de regiones modificado. La codificación JBIG una de las razones por la que no se usó fue porque necesitaba un número constante de bits de error para que funcionará correctamente. El codificador aritmético por otra parte no es influido por este hecho, aunque también por la buena respuesta de este tipo de codificadores se eligió. Con el crecimiento de regiones con número de bits variable se ha intentado conseguir reducir el rango dinámico de la *imagen de error*, pero hay otras dos representaciones de los datos existentes que son la *matriz de discontinuidades* y el *vector de bits altos de semilla*. Está claro que se reduce el rango dinámico de la imagen error. El vector de bits altos de semilla será el mismo que el de casos anteriores, pero la matriz de discontinuidades se cargará de más datos de

los que poseía anteriormente por lo que los resultados obtenidos no son tan buenos como podíamos pensar que ocurriera por un crecimiento de regiones con número de bits de error variable. Por la razón de que comprime menos el algoritmo el crecimiento de regiones modificado no se usará de aquí en adelante.

A continuación vamos a introducir en el esquema un bloque existente en el algoritmo de compresión JPEG-LS con el que vamos a intentar reducir la redundancia interpíxel de la imagen.

6.6 ALGORITMO DE COMPRESIÓN DE IMÁGENES BASADO EN SEGMENTACIÓN CON CRECIMIENTO DE REGIONES DE VECINDAD 4, DECORRELADOR Y CODIFICACIÓN ARITMÉTICA

En este nuevo esquema de compresión que presentamos, hemos prescindido del crecimiento de regiones modificado para que el número de bits de error fuera variable, por sus resultados de compresión. Por lo tanto lo que volvemos a usar es el crecimiento de regiones normal de vecindad 4 que por ser el mismo que el explicado en las versiones anteriores no volveremos a comentar detalladamente. La transformación $Y_sI_sQ_s$ que como dijimos, al reducir la redundancia intrapíxel de la imagen la seguiremos usando. El codificador que usaremos será el mismo codificador aritmético usado en las anteriores versiones sin ninguna modificación. Como siempre exponemos un esquema de bloques en la Figura 6-28.

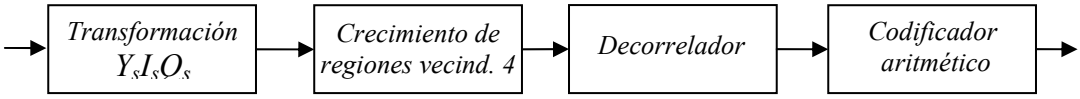


Figura 6-28 : Esquema de compresión basado en transformación $Y_sI_sQ_s$, crecimiento de regiones con vecindad 4,.decorrelador y codificación aritmética.

El nuevo elemento que tenemos en este esquema es el *decorrelador*. La posición de éste será objeto de estudio y se aclarará más delante. Un decorrelador reduce la correlación entre los píxeles vecinos, es un reductor de redundancia interpíxel. También lo llamaremos *predictor* en alguna ocasión.

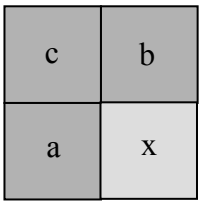
A continuación vamos a explicar el funcionamiento detallado de este decorrelador y presentar las dos posiciones donde éste se podría ubicar en el esquema de compresión y sus efectos en ambas posiciones.

6.6.1 Decorrelador. Predicción fija por filtro de mediana

Con este bloque se intenta reducir la redundancia interpíxel existente en una imagen. El crecimiento de regiones también efectúa una reducción de la redundancia entre los píxeles porque se encarga de almacenar la diferencia entre píxeles. Pues bien, ahora se va a intentar reducir aún más esa redundancia con este elemento.

En los esquemas generales de compresión sin perdidas existe un predictor. Este decorrelador que usamos en este esquema de compresión, es el predictor fijo que usa el algoritmo JPEG-LS[8]. El algoritmo JPEG-LS esta compuesto de un predictor con una componente fija y una variable, en este caso hemos usado sólo la componente fija que se le llama *filtro de mediana*. [Weinberger, 00]

El filtro de mediana es una función no lineal que posee una capacidad rudimentaria de detección de bordes. El funcionamiento del decorrelador lo podemos ver en la Figura 6-29.



$$x_{med} = \begin{cases} \min(a,b) & si \;\; c > \max(a,b) \\ \max(a,b) & si \;\; c \le \min(a,b) \\ a+b-c & en\ otro\ caso \end{cases}$$

Figura 6-29 : Función no lineal del filtro de mediana

Este filtro de mediana va pasando por toda la imagen de izquierda a derecha y de arriba abajo. En los casos en que alguno de los píxeles no sean accesibles, por ejemplo cuando estamos en un borde lateral y el píxel *a* y *c* no existen, entonces se usan sólo los que existan en cada momento. Una vez que hemos calculado el valor x_{med} para el píxel, al ser éste el valor predicho, se ha de restar al valor original del píxel y éste será el valor de la nueva imagen decorrelada. Comentar que el filtro se aplica primeramente a la imagen y después se van restando cada uno de los valores, porque de otro modo, es decir, calculando la salida del filtro por la imagen modificada, estaríamos haciendo mal el proceso, porque no aplicaríamos el predictor a la imagen.

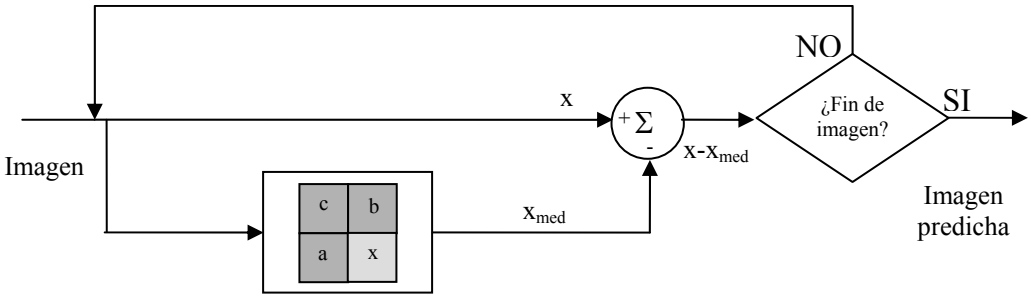


Figura 6-30 : Esquema descriptivo del predicor

Hay dos puntos donde podemos incluir el predictor, uno de ellos es detrás del crecimiento de regiones, y otro es previo a éste. En la Figura 6-31 y Figura 6-32 vemos gráficamente dichos puntos y a continuación explicamos sobre qué actúa el predictor. En las figuras vamos a omitir la transformación $Y_sI_sQ_s$.

[8] JPEG-LS son las siglas de Joint Photographic Experts Group-Lossless es el compresor sin perdidas de JPEG

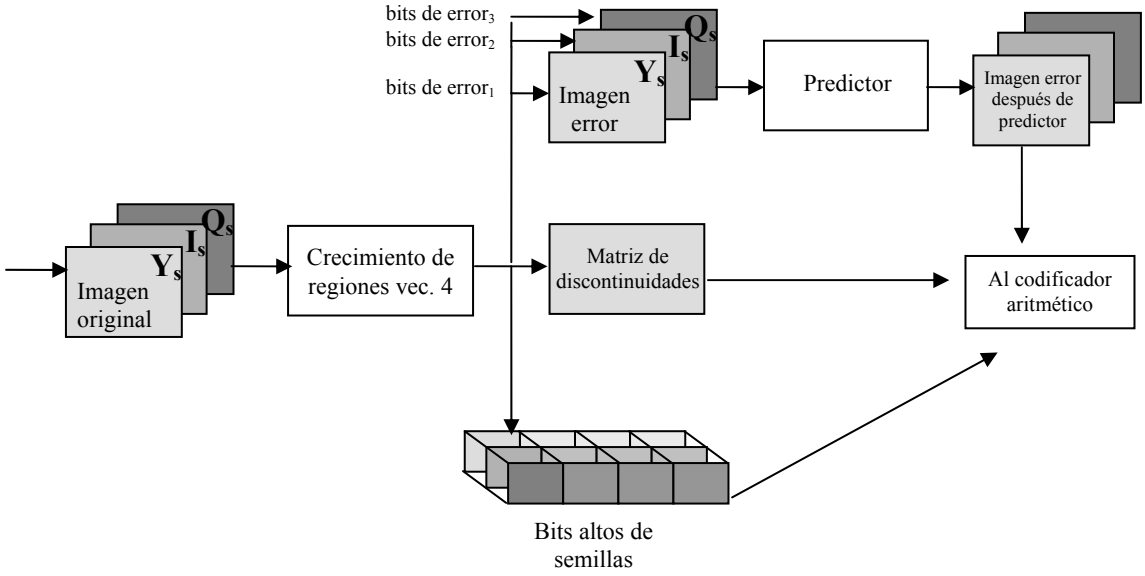


Figura 6-31 : Esquema de compresión en el que el predictor se coloca después de segmentar la imagen

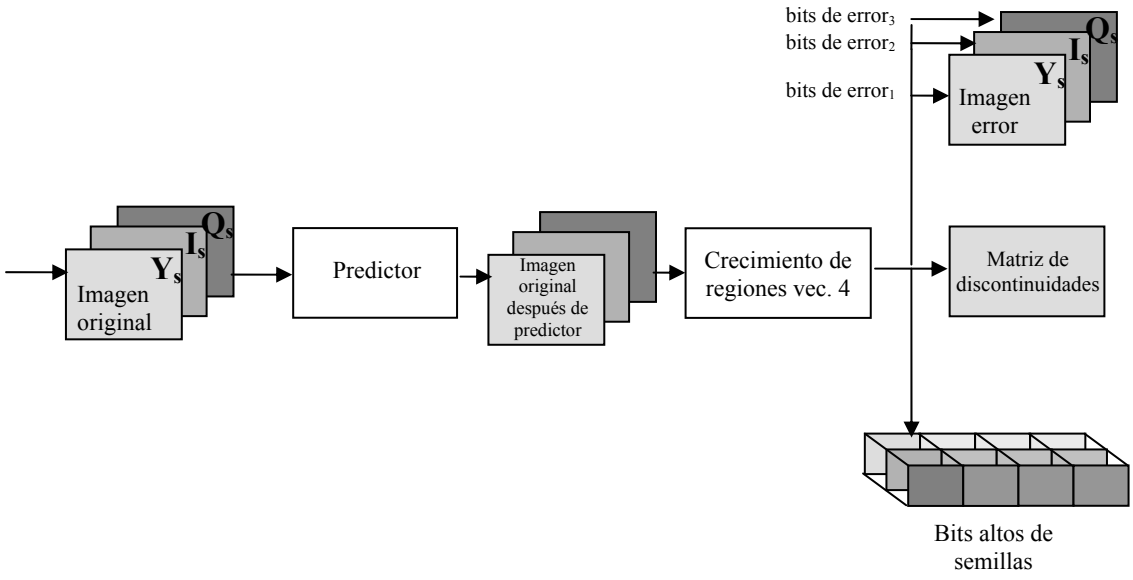


Figura 6-32 : Esquema de compresión en el que el predictor se coloca antes de segmentar la imagen

La primera posición que puede adoptar el predictor es detrás del crecimiento de regiones. El predictor sólo actúa sobre la imagen error en este caso, ya que es la única cuya información puede tener redundancias interpíxel, ya que la matriz de

discontinuidades y el vector bits altos de la semilla son una forma de representación de información que no posee dicha redundancia interpíxel.

La segunda posición es antes del crecimiento de regiones. Aquí como se verá en los resultados va a ocurrir que como el predictor hace la función de decorrelador de la imagen, al crecer posteriormente la imagen, va a estar decorrelada y el crecimiento de regiones no va a ser para nada efectivo porque las relaciones entre los píxeles vecinos han disminuido.

En ambos casos la imagen error decorrelada o no, dependiendo de la opción, la matriz de discontinuidades y el vector de bits altos de semilla se codifican aritméticamente con el mismo codificador usado hasta ahora.

El proceso de descodificación es igual que los algoritmos anteriores, la única diferencia es que en el punto donde hayamos incluido el predictor habrá que hacer el proceso inverso, en vez de restar el valor predicho se sumará.

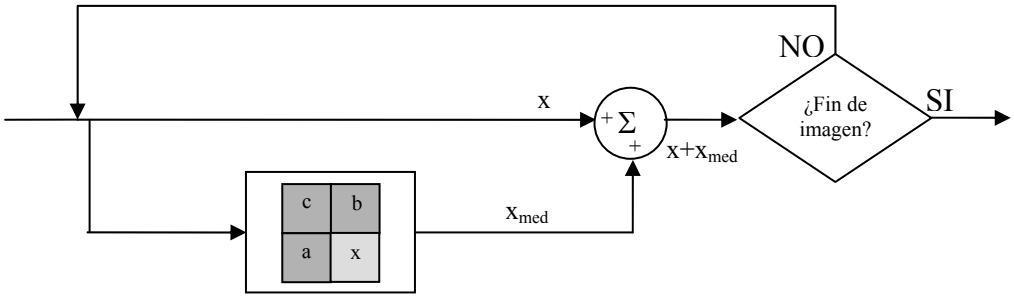


Figura 6-33 : Esquema del predictor en el decodificador, la única diferencia es que la cantidad predicha se suma

6.6.2 Resumen

En este esquema de compresión se ha incluido un nuevo bloque al algoritmo de compresión, estamos hablando del predictor. Con este bloque lo que hemos intentado hacer es reducir la redundancia interpíxel, para ellos hemos expuesto dos esquemas, uno con el predictor actuando sobre la imagen error, esto es, después del crecimiento, y otra antes del crecimiento. La segunda opción según se verá en los resultados no dio muy buenos resultados. La razón de que no funcione muy bien un predictor antes de un crecimiento de regiones es que, el predictor decorrela la imagen y esta redundancia interpíxel es necesaria para el crecimiento de regiones, por lo que si se reduce, el crecimiento de regiones no será efectivo. La opción de colocar el predictor después del crecimiento a la imagen error mejoró bastante la compresión.

El siguiente paso es mejorar el funcionamiento del codificador aritmético introduciendo contextos. Puesto que el número de regiones que salían de la segmentación no era muy grande, se pensó en suprimir el crecimiento y dejar solo al predictor. Los resultados obtenidos de esta forma fueron favorables, por lo que a partir de ahora el reductor de la redundancia interpíxel será sólo el predictor.

6.7 ALGORITMO DE COMPRESIÓN DE IMÁGENES CON TRANSFORMACIÓN $Y_SI_SQ_S$, PREDICTOR DE FILTRO MEDIANA Y CODIFICACIÓN ARITMÉTICA CON CONTEXTO

Este esquema es un poco diferente al que hemos ido siguiendo, en este caso no vamos utilizar el crecimiento de regiones. El dejar de usar la segmentación viene dado por los resultados obtenidos que se verán en el capítulo de resultados, que dan mejores tasas de compresor al predictor del algoritmo JPEG-LS que hemos usado aquí, aunque no en su totalidad, no es exactamente el mismo. Otra modificación importante será la inclusión del contexto en la codificación aritmética. Introduciremos dos tipos de calcular el contexto y en el capítulo de resultados se compararán. Por supuesto se sigue usando el espacio de color $Y_sI_sQ_s$.

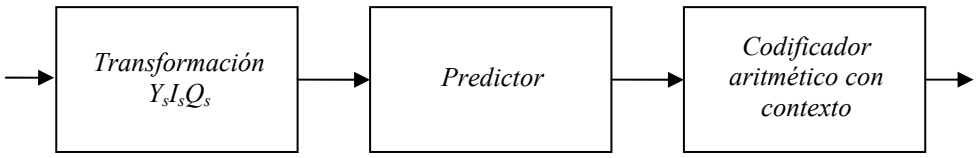


Figura 6-34 : Esquema de compresión basado en transformación $Y_sI_sQ_s$,predictor y codificación aritmética con contexto

6.7.1 Codificador aritmético con contexto normal

El buen funcionamiento de un codificador aritmético depende de en qué medida la distribución estadística real de los datos se adecua a la predicha, que es la que se emplea para el diseño del codificador. Cuanto más se aproxime la distribución estadística de los datos al modelo que se emplea en el codificador, el codificador aritmético se aproximará a la entropía de la fuente. Si usamos diferentes distribuciones estadísticas, obtendremos diferentes tasas de compresión. Además, cuanto menor sea la varianza de los datos mayor será la capacidad de compresión del codificador aritmético. Al codificar un píxel en un codificador aritmético normal, el modelo se adapta a los píxeles que se han codificado hasta el momento. En un codificador aritmético con contexto, el modelo se inicializa para cada grupo de datos pertenecientes a un mismo contexto. Estos datos poseerán menor desviación estadística [Triantafyllidis, 99].

Se divide el conjunto de datos a codificar en contextos donde, previsiblemente, la varianza de los datos será menor. Este reagrupamiento de píxeles permitirá que las tasas de compresión del codificador mejoren.

En la codificación con contexto normal se han definido 8 contextos que a partir de la imagen predicha se ha ponderado los píxeles vecinos para determinar en el contexto en que se encuentra el píxel que vamos a codificar. La ponderación se puede ver sobre que píxeles se produce en la Figura 6-35.

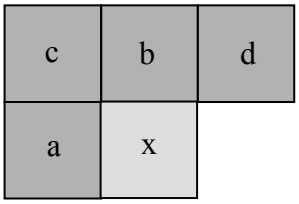


Figura 6-35 : Conjunto de píxeles que se van a usar en la predicción del contexto

Lo primero que se hace es, sobre la imagen predicha, calcular para cada píxel a qué contexto pertenecerá. Además habrá que almacenar el número de píxeles que pertenecen a cada contexto para la posterior decodificación. El contexto será determinado por un valor, resultado de la ponderación de los píxeles *a*, *b*, *c* y *d*.

$$x = \frac{a+b+c+d}{4}$$

Una vez calculado el valor ponderado de los píxeles, por medio de la Figura 6-36 se asignan los contextos.

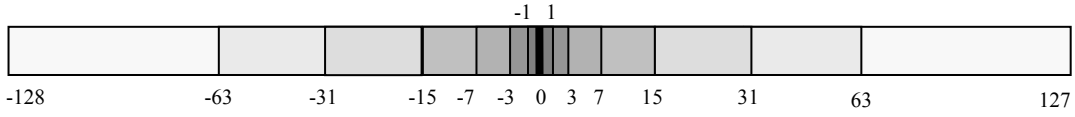


Figura 6-36 : Representación en escala de grises de los 8 contextos y que valores corresponden a cada uno

También podemos tener está información de asignación de los contextos mediante la Tabla 6-4.

Contexto	*Valores asignados*
Contexto 1	0
Contexto 2	-1,1
Contexto 3	-3,-2,2,3
Contexto 4	-7,...,-4,4,…,7
Contexto 5	-15,…,-8,8,…,15
Contexto 6	-31,…,-16,16,…,31
Contexto 7	-63,…,-32,32,…,63
Contexto 8	-128,…,-64,64,…127

Tabla 6-4 : Tabla que índica en que contexto se codifican cada uno de los valores.

Hay que decir que, aunque los valores que indican el contexto en el que se enmarca un píxel puedan ser negativos, los valores que hemos usado en los algoritmos son sin signo. Pero como nosotros usamos enteros sin signo un -1 será un 255, que a la hora de hacer algún cálculo, habrá que tener en cuenta que a pesar de ser 255 no puede predecir un contexto con ese número, sino con el que tendría con signo, esto es, -1.

Una vez hecho esto se sitúan todos los contextos en una tabla de tres veces el número de píxeles uno detrás de otro. Como ya no tenemos segmentación no necesitamos la matriz de discontinuidades. Se codificarán aritméticamente la imagen predicha y la matriz signo resultante de la transformación $Y_s I_s Q_s$. Cada vez que se codifique un contexto se inicializa el modelo del codificador aritmético, o si entra la matriz signo en el codificador.

Es necesario almacenar también en el archivo final de la imagen comprimida, el número de píxeles que componen cada contexto. Esto supondrá, si tenemos 8 contextos y suponiendo que sean 4 bytes para representar el número de elementos de cada contexto, un total de 32 bytes que a pesar que incrementa el tamaño del archivo final, las mejoras que introducen la codificación aritmética con contexto permiten su inclusión.

Lo único que vamos a codificar es la imagen predicha y la matriz signo, podemos ver esto esquemáticamente.

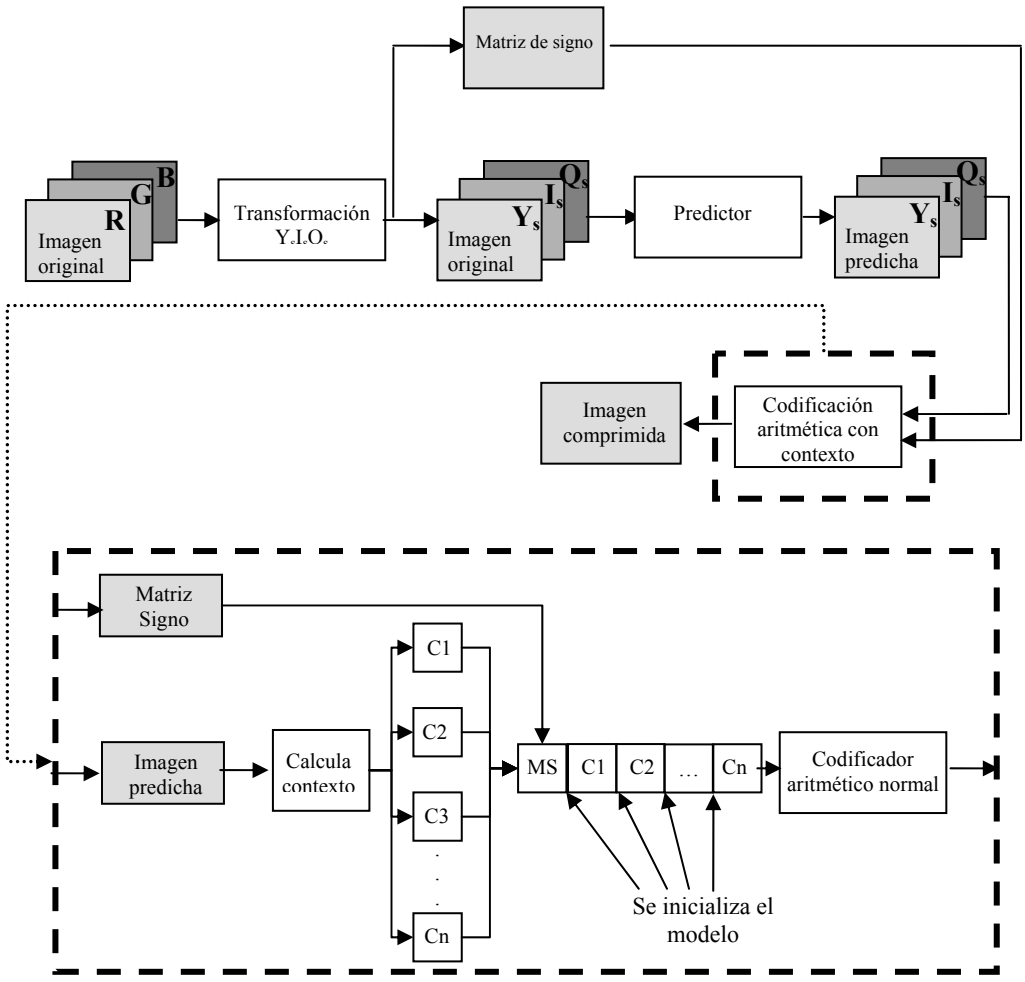


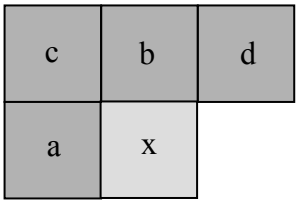
Figura 6-37: Esquema detallado de compresión predictiva con codificación aritmética con contexto. Se detalla el codificador aritmético con contexto.

En la descompresión de la imagen como dijimos antes es necesario el número de elementos de cada contexto, porque los datos saldrán decodificados pero no se sabrá a qué parte de la imagen pertenecen. Cuando se aplique de nuevo el bloque que calcula el contexto, se debería saber, después de conocer el contexto al que pertenece el siguiente píxel a procesar, dónde se encuentra ese píxel dentro de la fila de todos los contextos y la matriz signo que ha dado como resultado la descodificación aritmética. El número de elementos de cada contexto actúa como puntero de ordenación, indicando en qué "tabla" tenemos que ir sacando los datos.

6.7.2 Codificador aritmético con contexto modificado

Una última modificación que se realizó al cálculo del contexto está basada en gradientes locales. El contexto donde se sitúa el píxel en estudio lo predecimos calculando los gradientes de su entorno. Éste método aplica una mejora importante a la hora de calcular de predecir el contexto. Al estar basado en gradientes locales y no en una ponderación, es muy sensible a bordes en el entorno del píxel en cuestión. Esta mejora en la predicción influirá positivamente en el codificador, ya que los valores de los píxeles predichos para llevarlos a un contexto u a otro serán muy aproximados a los valores reales.

Los gradientes locales serán teniendo en cuenta los píxeles de su alrededor los que siguen:



$$g_1 = b - c$$
$$g_2 = d - b$$
$$g_3 = a - c$$

Esto tres valores muestran la actividad de la zona del píxel y pueden entenderse como si se tratase de derivadas, por lo tanto pueden predecir el valor de *x*. El valor de *x* es la suma de c más el gradiente vertical y el gradiente horizontal, teniendo en cuenta que los gradientes horizontales se ponderan.

$$x = c + \frac{g_1 + g_2}{2} + g_3$$

6.7.3 Resumen

En este último algoritmo no hemos usado segmentación, pero sí un predictor. También un elemento muy importante ha sido el codificador aritmético que habíamos

usado antes, pero modificado para que funcione con contexto. Todos estos elementos han llevado a este esquema al que mejor tasa de compresión ha alcanzado como se verá en los resultados, incluso superando al estándar de compresión sin pérdidas más conocido, el JPEG-LS.

CAPÍTULO 7

RESULTADOS COMPARATIVOS Y ANÁLISIS DE PRESTACIONES

Para analizar el comportamiento de los distintos algoritmos presentados se usaron 12 imágenes. Estaban divididas en tres categorías, con cuatro imágenes por categoría. Cuatro de ellas corresponden a imágenes médicas, más concretamente a imágenes de quemados que se les va a denominar por: *Foto1*, *Foto2*, *Foto3*, *Foto4*. La segunda categoría la podemos definir como categoría de imágenes generales o naturales que son *Alemania*, *Lena*, *Loros* y *Pimientos*. Y por último tenemos cuatro imágenes fractales que se denominan: *Mand*, *Yinywthm*, *7thm*, *Atlantis*. Se han elegido imágenes de distinta naturaleza para probar el algoritmo sobre imágenes de distintas características. Las imágenes médicas se caracterizan porque poseen baja información en alta frecuencia, mientras que en las imágenes generales abundan los detalles y cambios de color. En el tercer grupo la mayoría de la información es de luminancia.

A lo largo de este capítulo se van a ir comparando los distintos algoritmos que se han desarrollado, entre ellos, así como compararlos con compresores como el estándar JBIG, el estándar más conocido de compresión sin pérdidas JPEG-LS, y con el algoritmo SLCIC.

A continuación vamos a mostrar las 12 imágenes que se usaron para la obtención de resultados.

7.1 IMÁGENES MÉDICAS

Figura 7-1 : Imágenes médicas. Fotografías de quemaduras. De izquierda a derecha y de arriba abajo tenemos *Foto1*, *Foto2*, *Foto3*, *Foto4*.

7.2 IMÁGENES GENERALES

Figura 7-2: Imágenes generales. De izquierda a derecha y arriba abajo tenemos: *Alemania*, *Lena*, *Loros*, *Pimientos*

7.3 IMÁGENES FRACTALES

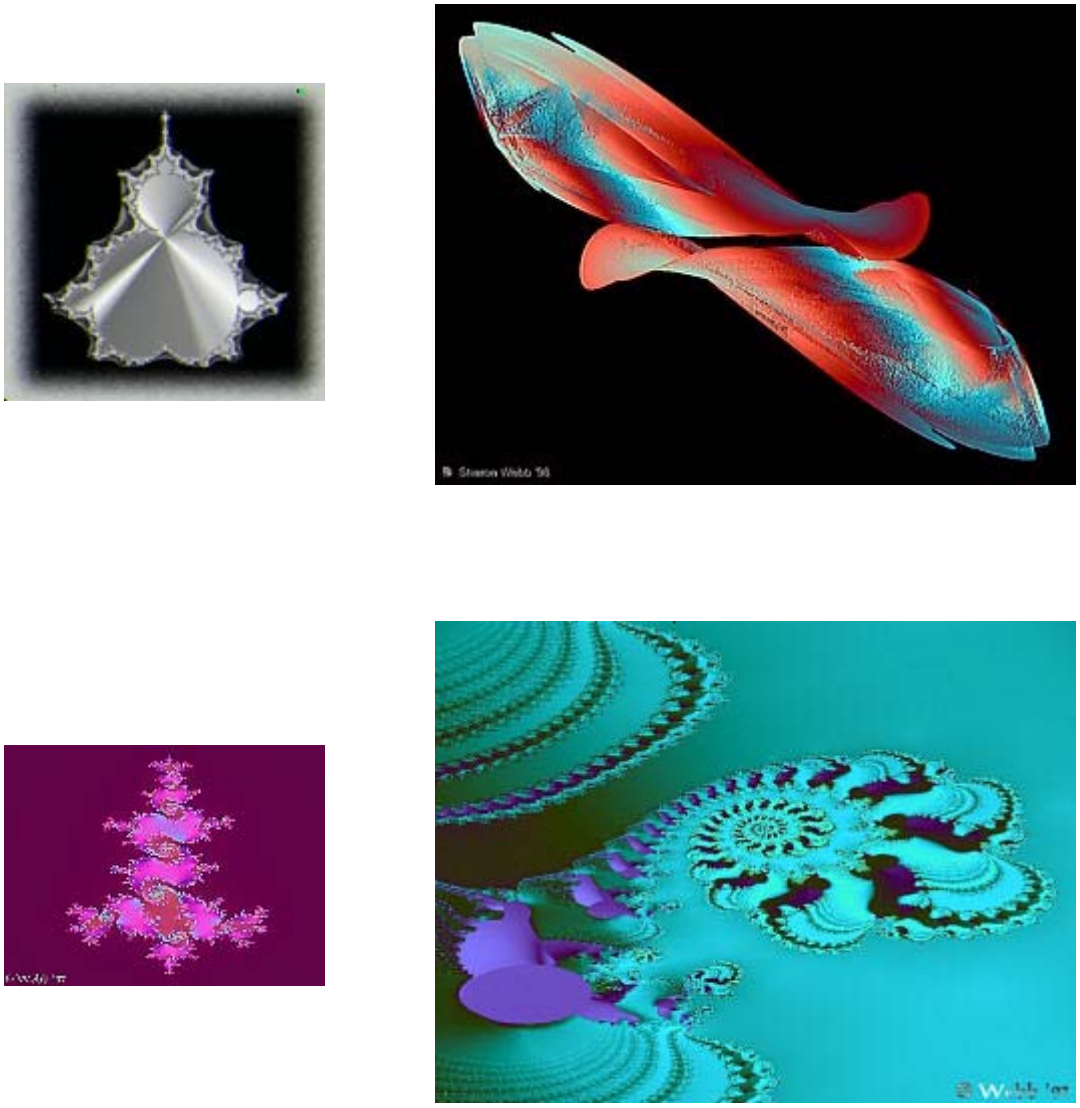


Figura 7-3 : Imágenes fractales. De izquierda a derecha y arriba abajo tenemos: *Mand*, *Yinywthm*, *7thm* y *Atlantis*

Las imágenes que vamos a usar para los resultados son todas en color real, esto es, que cada píxel está representado por 24 bits/píxel, 8 bits para cada componente de color. La resolución de cada una de las imágenes la mostramos en la Tabla 7-1. El formato[9] que poseen todas las imágenes es BMP, en su versión sin compresión, y para cada color R, G, B tenemos 8 bits.

Imágenes	*Resolución (columnas × filas)*
Fotografías de quemaduras	
Foto 1	832×624
Foto 2	832×624
Foto 3	832×624
Foto 4	832×624
Imágenes generales	
Alemania	768×512
Lena	512×512
Loros	768×512
Pimientos	512×512
Imágenes fractales	
Mand	160×159
Yinywthm	320×240
7thm	160×120
Atlantis	320×240

Tabla 7-1: Resolución de cada una de las imágenes del estudio.

El primer resultado que obtuvimos fue concerniente al formato BMP (véase Apéndice A). Partíamos de imágenes BMP en las que el formato del conjunto de imágenes fractales, difería de los conjuntos de médicas y generales. Su variación consistía en el *header* del fichero BMP. Más concretamente consistía en la existencia, en el de las fractales, de 4 bytes con valor 0 al comienzo del bitmap, o lo que es lo mismo, al final del header. Inicialmente usábamos un lector de imágenes BMP que no tenía en cuenta este hecho. Al no tener en cuenta esto, las imágenes al ser descomprimidas, veían cambiados sus planos de color (las fractales). Una primera posibilidad que se barajó fue la de cambiar en el código a la hora de comprimir las imágenes fractales añadiendo 4 bytes de lectura antes de llegar a el bimap. La posibilidad era válida pero habría que modificar el código fuente cada vez que quisiéramos comprimir imágenes de un *header* diferente al más común. Viendo que ésta posibilidad no era adecuada, intentamos buscar información acerca de este hecho. La mayoría de los documentos consultados no hacían referencia a este hecho, el de que hubiera distintos formatos BMP. Debido a la falta de información por algún bit o byte que indicara el comienzo del bitmap, fuimos comparando dos imágenes, una fractal y otra general. Cada uno de los campos de la cabecera lo comprobamos con las características de la imagen, esto es, resolución horizontal, vertical… hasta que comprobamos que un dword (0Ah-0Dh) , del cual no quedaba claro su funcionamiento,

[9] El formato de estas imágenes es BMP, pero hay diferencia en los formatos de algunas imágenes que se explicará a continuación.

variaba su valor. Después de pasar este valor a decimal, teniendo en cuenta el formato Intel, observamos que dicho valor indicaba el tamaño que tenia la cabecera completa, esto es, desde que comienza el archivo hasta que empieza el bitmap. Pues bien, programando el algoritmo correspondiente llevamos el lector de imágenes a un nivel un poco más automático de lo que era anteriormente.

7.4 ALGORITMO DE DECORRELACIÓN INTRAPÍXEL. TRANSFORMADA $Y_SI_SQ_S$

Para la reducción de la redundancia intrapíxel, nos hemos movido del espacio de color RGB al espacio de color $Y_sI_sQ_s$ en el que la mayor parte de información de alta frecuencia se sitúa en la componente *Y* y la parte de crominancia son las componentes *I* y *Q*, que a su vez son ortogonales. La transformación que hemos usado ha sido una transformación entera, ya que estamos hablando de un algoritmo de compresión sin pérdidas, y no es posible una transformación en punto fijo por las pérdidas que introduciría. Ésta ha sido la transformación que hemos estado usando en todos los algoritmos que se han analizado, por lo que presentamos la Tabla 7-2 en la que se observa cómo la entropía de la imagen se concentra en la componente de luminosidad, lo que se aprovecha en los algoritmos de compresión. El cálculo de la entropía de la imagen ha sido una estimación de orden 1.

Imágenes	Y_s	I_s	Q_s
Foto 1	6,9	5,2	4,3
Foto 2	7,1	5,3	3,1
Foto 3	6,1	5,2	4,3
Foto 4	6,5	5,1	4,2
Media	**6,6**	**5,2**	**4,0**
Alemania	7,7	5,6	4,3
Lena	7,4	6,1	4,9
Loros	7,2	6,4	5,9
Pimientos	7,6	6,5	6,2
Media	**7,5**	**6,1**	**5,3**
Mand	6,5	1,2	1,1
Yinywthm	3,8	3,7	3,2
7thm	4,1	3,7	3,8
Atlantis	7,2	6,6	4,9
Media	**5,4**	**3,8**	**3,2**

Tabla 7-2: Tabla de la entropía de cada componente YIQ de diferentes imágenes.

Como podemos observar, la componente *Y* tiene siempre un valor más alto que las demás, esto indica por medio del cálculo de la entropía que existe más información en la componente *Y* que en las demás. Este hecho vamos a contraponerlo al cálculo de la entropía en las mismas imágenes pero en el espacio de color RGB en la Tabla 7-3.

Imágenes	R	G	B
Foto 1	6.8	6,8	7,2
Foto 2	7,5	7,1	6,6
Foto 3	6,4	6,3	6,5
Foto 4	6,7	6,7	6,4
Media	**6,9**	**6,7**	**6,7**
Alemania	7,6	7,7	7,6
Lena	7,3	7,6	7,0
Loros	7,5	7,5	7,2
Pimientos	7,3	7,5	7,1
Media	**7,4**	**7,6**	**7,2**
Mand	6,5	6,4	6,5
Yinywthm	3,8	3,6	4,0
7thm	4,4	4,2	4,9
Atlantis	7,7	7,5	6,2
Media	**5,6**	**5,4**	**5,4**

Tabla 7-3 : Tabla de la entropía de cada componente RGB de diferentes imágenes.

En este caso observamos que todas las componentes tienen la misma información, es decir, que una imagen en el espacio de color RGB tiene redundancia debido a que una misma información va en los tres planos de color. En las siguientes fotografías que mostramos, hemos tomado una imagen de prueba Figura 7-4 y la hemos dividido en sus tres planos de color R, G y B.

Figura 7-4 : Foto de Lena en el espacio de color RGB

Figura 7-5 : Las intensidades de las tres componentes R (izquierda), G(centro) y B(derecha) de la imagen de Lena

Como podemos observar y como anunciaba la tabla de la entropía de cada una de las componente de la imagen, las tres componentes tienen una alta redundancia entre los tres planos de color porque en las tres se puede apreciar claramente la imagen original. Esta redundancia entre los tres planos hace que si comprimiéramos sin transformación de espacio de color, estaríamos comprimiendo información de más que se puede evitar. La transformación $Y_sI_sQ_s$ intenta disminuir esa correlación entre las tres componentes del espacio de color. Podemos ver el resultado que se obtiene transformando la imagen en la Figura 7-6.

Figura 7-6 : Las intensidades de las tres componentes Y (izquierda), I(centro) y Q(derecha) de la imagen de Lena

En este caso como podemos comprobar la componente *Y* tiene más información en alta frecuencia que las otras dos. En las componentes de crominancia no se distingue bien la imagen. Esto corrobora los valores menores de entropía para las componentes de crominancia, como se obtenían en las tablas. Con esto hemos demostrado que con esta transformación $Y_sI_sQ_s$ hemos reducido la redundancia intrapíxel, lo cual producirá una ganancia de compresión al completar el proceso de compresión que utilicemos.

Vamos a hacer una comparación del mismo algoritmo de compresión, uno cuyos resultados se expondrán a continuación, sin realizar el proceso de cambio de espacio de color y otro en el que se realiza.

Imágenes	*Algoritmo sin transformación* $Y_sI_sQ_s$	*Algoritmo con transformación* $Y_sI_sQ_s$
Fotografías de quemaduras		
Foto 1	10,00	6,83
Foto 2	6,66	5,49
Foto 3	6,77	5,46
Foto 4	6,87	5,54
Media	**7,75**	**5,83**
Imágenes generales		
Alemania	16,49	11,80
Lena	14,50	14,24
Loros	11,64	9,60
Pimientos	15,59	15,78
Media	**14,56**	**12,86**
Imágenes fractales		
Mand	13,06	8,67
Yinywthm	7,76	7,87
7thm	10,72	10,35
Atlantis	12,99	12,38
Media	**11,14**	**9,82**
Media Global	**11,15**	**9,50**

Tabla 7-4 : Comparación de un algoritmo que se expondrá posteriormente sin transformación de espacio de color y el mismo algoritmo pero con transformación de color

Como observamos en la Tabla 7-4 la reducción de la redundancia intrapíxel queda patente a la vista de estos datos, se consigue hasta una mejora de 1,92 bits/píxel en las imágenes médicas y un 1,7 bits/píxel en las imágenes generales. Las imágenes fractales mejoran en 1,32 bits/píxel. Por lo tanto queda demostrada la utilización de la transformación $Y_sI_sQ_s$.

7.5 RESULTADOS OBTENIDOS EN LOS DIFERENTES ALGORITMOS

A continuación vamos a exponer todos los resultados obtenidos para cada uno de los algoritmos desarrollados. El primer estudio que se realiza con los algoritmos de

compresión sin pérdidas basados en segmentación es comparar el algoritmo SLCIC[10] que tiene crecimiento de regiones con vecindad 4, con otro que posea vecindad 8 siempre manteniendo todo lo demás, esto es, el mismo codificador, la misma transformación de espacio de color…. La idea de comparar estas dos formas de crecimiento de regiones es debida a que se quiere obtener un referente a la hora de aplicar segmentación a compresión. Este referente será importante cuando apliquemos los algoritmos sucesivos, ya que se elegirá la que mejores resultados obtenga.

Lo que se ha intentado en esta comparación es experimentalmente elegir entre uno de los dos tipos de crecimiento. Para ellos hemos usado las 12 imágenes de prueba. Además de la compresión, que la expresaremos en bits/píxel, calcularemos el número de regiones generadas por uno y otro método.

Imágenes	*Algoritmo Vecindad 4 (SLCIC)*	*Algoritmo Vecindad 8*
Fotografías de quemaduras		
Foto 1	5,85	6,90
Foto 2	4,02	5,10
Foto 3	4,21	5,19
Foto 4	4,30	5,24
Media	**4,60**	**5,61**
Imágenes generales		
Alemania	12,34	13,43
Lena	14,98	15,49
Loros	10,10	10,10
Pimientos	16,33	16,57
Media	**13,44**	**13,90**
Imágenes fractales		
Mand	7,72	8,25
Yinywthm	8,84	8,70
7thm	11,00	11,49
Atlantis	12,76	13,58
Media	**10,08**	**10.50**
Media Global	**9,38**	**10,00**

Tabla 7-5: Comparación de las tasas de compresión del algoritmo de compresión con vecindad 4 y 8. El resto de elementos en los dos algoritmos es el mismo, transformación $Y_sI_sQ_s$ y el codificador JBIG. Los valores están expresados en bits/píxel

Observando la Tabla 7-5 por grupos de imágenes vemos que en el grupo de imágenes de quemados, es donde la diferencia entre los dos métodos de crecimiento de regiones se acentúa más, en media la diferencia es de 1,01 bits/píxel. En este tipo de imágenes el incremento es demasiado grande. Para los otros dos casos de imágenes, las generales y las fractales, la diferencia a favor del crecimiento de vecindad 4 es

[10] SLCIC son las siglas de Segmentation-based Lossless Color Image Compression y es un algoritmo que posee una transformación $Y_sI_sQ_s$, crecimiento de regiones con vecindad 4 y codificador JBIG.

0,46bits/píxel y 0,42bits/píxel respectivamente. Aunque estos valores puedan resultar pequeños, la diferencia de una imagen de 768×512 de 393.216 píxeles comprimidas con uno u otro método llega a (usando el valor de 0.46bits/píxel) 22609 bytes. Cantidad esta anterior importante si estamos hablando de compresión sin pérdidas, porque la calidad se mantiene.

En cuanto al número de regiones que se crecen con uno y otro algoritmo, en el único grupo que hay diferencia es el de las imágenes fractales como vemos en Tabla 7-6.

Imágenes	*Algoritmo Vecindad 4*	*Algoritmo Vecindad 8*
Imágenes fractales		
Mand	72	26
Yinywthm	11625	8465
7thm	2463	2133
Atlantis	200	3

Tabla 7-6: Comparación para las imágenes fractales del número de regiones crecidas en los dos algoritmos.

Hablando en términos de media global la diferencia que compara a los dos algoritmos es de 0,62 bits/píxel, por lo que de ahora en adelante cualquier algoritmo que usemos para segmentación será de vecindad 4, como ha demostrado su mejor comportamiento frente al de 8, teniendo todos los demás bloque las mismas características.

7.5.1 Codificador aritmético

Una vez comprobado que el crecimiento de regiones de vecindad 4 es más efectivo que el de 8, el siguiente paso es cambiar el codificador. Pasamos de uno JBIG a uno aritmético, con el que podremos comparar el funcionamiento conjunto de la segmentación y del codificador aritmético. Después de haber demostrado con el algoritmo SLCIC que la segmentación con el codificador JBIG obtienen resultados muy buenos, sobre todo para aplicaciones médicas, lo siguiente que vamos a comparar es la compresión del algoritmo SLCIC y el que resulta de cambiar el codificador JBIG por el aritmético, en las siguiente Tabla 7-7 tenemos los resultados.

Imágenes	*Algoritmo Vecindad 4 y codificación JBIG(SLCIC)*	*Algoritmo Vecindad 4 y codificación aritmética*
Fotografías de quemaduras		
Foto 1	5,85	8,86
Foto 2	4,02	7,15
Foto 3	4,21	7,43
Foto 4	4,30	7,57
Media	**4,60**	**7,75**
Imágenes generales		
Alemania	12,34	13,23
Lena	14,98	14,36
Loros	10,10	10,23
Pimientos	16,33	15,79
Media	**13,44**	**13,40**
Imágenes fractales		
Mand	7,72	9,00
Yinywthm	8,84	11,14
7thm	11,00	12,64
Atlantis	12,76	15,14
Media	**10,08**	**11,98**
Media Global	**9,38**	**11,04**

Tabla 7-7: Comparación de las tasas de compresión del algoritmo de compresión con vecindad 4 con codificador JBIG y otro aritmético. Los valores están expresados en bits/píxel

Como podemos observar en la Tabla 7-7en el caso de las imágenes médicas los resultado obtenidos son peores usando el codificador aritmético, y estas diferencias alcanzan de media 3.15 bits/píxel lo que no es tolerable en ningún caso ya que es un incremento de tamaño de la imagen muy grande. El único grupo de imágenes que mejora el algoritmo que usa codificador aritmético al que usa el codificador JBIG, es el grupo de las generales donde la media de éstas es 0.04 bits/píxel mejor para el aritmético. En el último grupo, el de imágenes fractales, la diferencia sigue siendo bastante grande como se puede apreciar.

El tamaño de las regiones generadas en cada imagen es el mismo para ambos algoritmos, esto es debido a que el algoritmo de crecimiento es el mismo en ambos casos, lo que varía es la forma de codificar la *imagen error* y la *matriz de discontinuidades* surgidas de la segmentación.

El tiempo de compresión/descompresión de estos dos algoritmos se puede comparar observando la siguiente Tabla 7-8.

Imágenes	*Algoritmo Vecindad 4 y codificación JBIG(SLCIC) (comp/descomp)*	*Algoritmo Vecindad 4 y codificación aritmética (comp/descomp)*
Fotografías de quemaduras		
Foto 1	1/1	4/1
Foto 2	1/1	4/1
Foto 3	1/1	4/1
Foto 4	1/1	4/1
Imágenes generales		
Alemania	1/1	3/1
Lena	1/1	2/1
Loros	1/1	3/1
Pimientos	1/1	2/1
Imágenes fractales		
Mand	1/1	1/1
Yinywthm	46/46	46/43
7thm	2/2	2/1
Atlantis	0/0	0/0

Tabla 7-8: Tiempos en segundos de compresión/descompresión de los algoritmos, uno con transformación $Y_sI_sQ_s$ crecimiento de regiones con vecindad 4 y codificador JBIG, y otro con transformación $Y_sI_sQ_s$ crecimiento de regiones con vecindad 4 y codificador aritmético.

El algoritmo que emplea la codificación JBIG obtiene mejores resultados en cuanto a tiempos de compresión, mientras que para los tiempos de descompresión ambos algoritmos poseen los mismos valores temporales.

Con esta comparación hemos constatado que el crecimiento de regiones normal no funciona muy bien en conjunto con el codificador aritmético, pero también hay que decir que el codificador JBIG posee un decorrelador previo a el codificador, por lo cual seguiremos usando el codificador aritmético e intentaremos reducir más la redundancia previa al codificador. La transformación de espacio de color $Y_sI_sQ_s$ va a ser una constante en todas las combinaciones que probemos, pero tenemos que intentar buscar otras fórmulas que mejoren el resultado con el codificador aritmético usado.

7.5.2 Crecimiento de regiones modificado

Una de las razones por las que se cambio de codificador fue el crecimiento de regiones modificado. En el caso del codificador JBIG, el crecimiento de regiones debía hacerse con un número constante de bits de error, por lo que si modificábamos el algoritmo de crecimiento de regiones para que el número de bits de error fuera variable, el codificador JBIG no podría usarse. En este caso vamos a comparar el esquema de compresión que incluye el crecimiento de regiones modificado. En este crecimiento como ya se explicó, se va variando el numero de bits de error, o lo que es lo mismo, el nivel de error que debe de haber entre un píxel y su central para que éste puede entrar en

la región, dependiendo si nos encontramos en una región de alta actividad o de baja. A continuación en la Tabla 7-9 vamos a comparar los valores de compresión obtenidos en el algoritmo de crecimiento normal con codificación aritmética y el algoritmo que posee crecimiento de regiones modificado y codificación aritmética. También iremos incluyendo la comparación con los estándares JBIG y JPEG-LS para tener una orientación de hasta qué nivel de compresión sería interesante llegar si fuera posible, claro.

Imágenes	*Crecimiento modificado y codificación aritmética*	*Crecimiento de vecindad 4 y codificación aritmética*	*SLCIC*[11]	*JPEG-LS*	*JBIG*
Fotografías de quemaduras					
Foto 1	9,05	8,86	5,85	9,65	12,43
Foto 2	7,14	7,15	4,02	6,37	8,19
Foto 3	7,28	7,43	4,21	6,68	8,68
Foto 4	7,50	7,57	4,30	6,57	8,67
Media	**7,74**	**7,75**	**4,60**	**7,32**	**9,49**
Imágenes generales					
Alemania	15,35	13,23	12,34	15,78	17,49
Lena	19,07	14,36	14,98	13,62	15,12
Loros	11,61	10,23	10,10	10,42	12,06
Pimientos	22,06	15,79	16,33	14,29	15,65
Media	**17,03**	**13,40**	**13,44**	**13,52**	**15,08**
Imágenes fractales					
Mand	9,00	9,00	7,72	9,70	11,27
Yinywthm	12,27	11,14	8,84	7,03	8,17
7thm	13,91	12,64	11,00	9,70	11,27
Atlantis	17,32	15,14	12,76	12,42	14,05
Media	**13,13**	**11,98**	**10,08**	**9,71**	**11,19**
Media Global	**12,63**	**11,04**	**9,38**	**10,35**	**12,19**

Tabla 7-9 : Comparación de los algoritmos de crecimiento de regiones modificado con el de crecimiento de vecindad 4 normal y con los estándares JPEG-LS y JBIG, además de SLCIC. Valores en bits/píxel

En la Tabla 7-9 vamos a comparar primeramente el algoritmo de crecimiento modificado con el de crecimiento normal. En cuanto al primer grupo de imágenes la compresión de uno y otro es prácticamente la misma, mientras que en los otros dos grupos existe una diferencia muy grande. Esto puede ser debido a la gran cantidad de detalles que poseen estas imágenes, que hagan que el algoritmo genere un número muy elevado de regiones, lo que se traduce en una matriz de discontinuidades muy cargada

[11] Este algoritmo de compresión es el que se desarrollo en [Serrano, 01], y que tiene una transformación de imagen $Y_sI_sQ_s$ un crecimiento de regiones de vecindad 4 y un codificador JBIG. Es el mismo que usamos en la primera comparación, que lo comparamos con el de vecindad 8

de información, que antes estaba más libre con el crecimiento normal de regiones de vecindad 4. Que un esquema de compresión contenga un bloque de segmentación de la imagen, no sigue que deba segmentar como un algoritmo puro de segmentación, sino que debe hacer una segmentación más suave, sin generar tal número de regiones.

Si comparamos el número de regiones que genera el algoritmo modificado de crecimiento de regiones, frente al de crecimiento normal, nos damos cuenta que la información necesaria para representar los bordes de las regiones es mucha.

Imágenes	*Crecimiento modificado y codificación aritmética*	*Crecimiento de vecindad 4 y codificación aritmética*
Fotografías de quemaduras		
Foto 1	2806	1
Foto 2	106	1
Foto 3	259	1
Foto 4	659	2
Imágenes generales		
Alemania	12512	1
Lena	21557	12
Loros	4369	1
Pimientos	17699	2
Imágenes fractales		
Mand	72	72
Yinywthm	16200	11625
7thm	3557	2463
Atlantis	6268	200

Tabla 7-10: Comparación para las imágenes del número de regiones crecidas en los algoritmos de crecimiento de regiones modificado y el crecimiento de regiones normal con vecindad 4.

Como podemos observar en la Tabla 7-10 el número de regiones que se crean con el algoritmo de crecimiento modificado es muy alto, y para codificar toda la información referente a los bordes de todas esas regiones se necesitarán muchos datos.

Podemos comparar también los tiempos de compresión del algoritmo con crecimiento modificado y aquel crecimiento normal de vecindad 4. Debido a la cantidad de regiones que se generan y de la complejidad del algoritmo, computacionalmente hablando (Tabla 7-11), los tiempos son bastante altos. Sin querer defender los altos tiempos de compresión, un algoritmo de compresión sin pérdidas, es para eso, para comprimir información importante que no debe sufrir transformación alguna, no para una transmisión en tiempo real, aunque sería muy interesante esa opción.

Imágenes	*Crecimiento modificado y codificación aritmética (comp/descomp)*	*Crecimiento de vecindad 4 y codificación aritmética (comp/descomp)*
Fotografías de quemaduras		
Foto 1	76/73	4/1
Foto 2	6/4	4/1
Foto 3	10/8	4/1
Foto 4	20/8	4/1
Imágenes generales		
Alemania	246/246	3/1
Lena	283/290	2/1
Loros	92/88	3/1
Pimientos	231/234	2/1
Imágenes fractales		
Mand	3/3	1/1
Yinywthm	61/58	46/43
7thm	3/2	2/1
Atlantis	20/22	0/0

Tabla 7-11: Tiempos en segundos de compresión/descompresión de los algoritmos, uno con transformación $Y_sI_sQ_s$ crecimiento de regiones modificado y codificador aritmético, y otro con transformación $Y_sI_sQ_s$ crecimiento de regiones normal con vecindad 4 y codificador aritmético.

Aunque el tiempo de compresión no sea una razón que excluya a un algoritmo, debe ser un parámetro importante a la hora de evaluar un algoritmo. Como hemos dicho, no buscamos un algoritmo de tiempo de compresión bajos, aunque se buscaría el más bajo entre los de las mismas prestaciones en cuanto a tasas, sino uno que posea una ganancia de compresión alta.

En la Tabla 7-9 de comparación con los estándares JPEG-LS y JBIG y el algoritmo SLCIC podemos observar cómo el algoritmo de crecimiento modificado y el de crecimiento normal para el caso de imágenes de quemados ya igualan la tasa de compresión del estándar JPEG-LS, y superan en compresión a JBIG, sin poder llegar a la altísima compresión que se alcanza con el algoritmo SLCIC [Serrano, 01] para este tipo de imágenes. Si comparamos los valores de crecimiento modificado, ocupa 3,25 bit/píxel más de media con respecto al que comprime más, el SLCIC. El número de bits/píxel que necesita el estándar JBIG es mayor a los demás, exceptuando al algoritmo de crecimiento modificado. En cuanto al grupo de imágenes fractales el algoritmo que obtiene el mejor valor es el JPEG-LS. Mejora en 3,42 bits/píxel al algoritmo de crecimiento modificado y en 2,27 al algoritmo de crecimiento normal de vecindad 4.

Una vez visto los resultados para los dos algoritmos que nos pueden marcar la pauta a seguir, hay que decidirse por qué línea seguir. El algoritmo de crecimiento modificado con codificador aritmético estaba pensado para ir variando el número de bits de error, que harían que un píxel vecino se incluyera o no en una región. La variación del número de bits de error buscaba reducir lo máximo el rango dinámico para

representar la *imagen error*, con la consiguiente ganancia que produciría eso, pero ocurre que el número de regiones en las que se divide la imagen es muy alta, por lo que la información de los bordes, representada en la matriz de discontinuidades, necesitará muchos datos para ser representada. Todo esto lleva a que cuando la imagen tiene muchos detalles como en las imágenes generales y las fractales la compresión disminuye.

Por los resultados obtenidos comparando estos dos algoritmos, para seguir aplicando mejoras al esquema de compresión, debemos de tomar aquél que mejores tasas de compresión ofrece. A partir de ahora se utilizará el algoritmo de crecimiento normal con vecindad 4 por tener mejor tasa de compresión que el modificado, y por tener un coste computacional mucho más bajo.

7.5.3 Decorrelador[12]

Después de elegir el algoritmo con el que vamos a seguir probando, es decir, el de crecimiento de regiones normal con vecindad 4, ahora vamos a intentar introducir otro bloque que disminuya algún tipo de correlación. Es el caso de la redundancia interpíxel. Vamos a usar un predictor (decorrelador) para disminuir esa redundancia. También vamos a ir observando los valores que obtenemos y comparándolos con los resultados de los estándares JPEG-LS y JBIG y con el compresor SLCIC.

Una cuestión que nos hacemos a la hora de colocar el decorrelador es si colocarlo previo o posterior al crecimiento de la imagen. Esta respuesta quedará justificada por las pruebas que vamos a realizar. Ni que decir tiene que todos los esquemas de compresión que estamos desarrollando tienen el bloque de transformada de espacio de color $Y_sI_sQ_s$, que como se demostró previamente proporciona una reducción de la redundancia intrapíxel.

A continuación vamos a mostrar en la Tabla 7-12 el algoritmo de crecimiento de regiones normal de vecindad 4, y a continuación pondremos el mismo algoritmo pero intercalando el bloque decorrelador delante y detrás del crecimiento y el que mejor resultado dé, lo compararemos con los demás algoritmos que nos sirven de base para la comparación.

[12] Usamos la palabra decorrelador pero indistintamente usaremos la palabra predictor. La finalidad es la de reducir la redundancia interpíxel.

Imágenes	*Crecimiento normal de vecindad 4 y codificación aritmética sin predictor*	*Crecimiento normal de vecindad 4, predictor y codificación aritmética*	*Predictor, crecimiento normal de vecindad 4 y codificación aritmética*
Fotografías de quemaduras			
Foto 1	8,86	7,40	--
Foto 2	7,15	6,12	--
Foto 3	7,43	6,10	--
Foto 4	7,57	6,22	--
Media	**7,75**	**6,46**	**--**
Imágenes generales			
Alemania	13,23	13,36	--
Lena	14,36	16,04	--
Loros	10,23	11,04	--
Pimientos	15,79	17,92	--
Media	**13,40**	**14,59**	**--**
Imágenes fractales			
Mand	9,00	10,39	--
Yinywthm	11,14	11,96	--
7thm	12,64	13,88	--
Atlantis	15,14	14,81	--
Media	**11,98**	**12,76**	**--**
Media Global	**11,04**	**11,27**	**--**

Tabla 7-12: Comparación del algoritmo de crecimiento normal con vecindad 4 con los algoritmos de crecimiento normal de vecindad 4, en la segunda columna colocando el predictor (decorrelador) posterior al crecimiento y en la tercera columna el predictor precede al crecimiento de regiones. Los valores están expresados en bits/píxel.

Lo primero a destacar de la Tabla 7-12 es que no obtenemos valores del esquema en el que el predictor precede al crecimiento de regiones. No pudimos obtener valores porque cuando se ejecutaba el programa que contenía este código tardaba muchísimo en comprimir las imágenes sin poder haber llegado a ver el resultado que se obtenía. Este tiempo tan alto se debe a que el crecimiento de regiones se estaba haciendo sobre una imagen que no era apta para la segmentación. La imposibilidad de realizar un crecimiento de regiones posterior a una decorrelación de la imagen puede intuirse si consideramos que el decorrelador lo que hace es eso, decorrelar la imagen, y al intentar crecer una imagen decorrelada que no mantiene ya la redundancia necesaria para ser crecida no se podría obtener resultado alguno. Una vez explicado esto vamos a comparar el efecto del predictor, con relación al esquema que no lo poseía.

Como podemos observar en la Tabla 7-13, las imágenes de quemados han ganado compresión con el predictor hasta 1,29 bits/píxel más. Esto es una mejora considerable. Se ha podido conseguir debido a que las imágenes de quemados son muy homogéneas, con lo que un predictor realizaría en gran medida bien su función. Por ejemplo en las imágenes naturales con más detalles, y por lo tanto con menos homogeneidad, el predictor empeora prácticamente lo mismo que se mejora en las imágenes médicas, 1,19 bits/píxel. En cuanto a las imágenes fractales se percibe un incremento en el número de bits/píxel necesarios para representar la imagen, y su valor es de 0,78 bits/píxel más que sin predictor.

Imágenes	*Crecimiento normal de vecindad 4 y codificación aritmética sin predictor*	*Crecimiento normal de vecindad 4, predictor y codificación aritmética*	*SLCIC*	*JPEG-LS*	*JBIG*
Fotografías de quemaduras					
Foto 1	8,86	7,40	5,85	9,65	12,43
Foto 2	7,15	6,12	4,02	6,37	8,19
Foto 3	7,43	6,10	4,21	6,68	8,68
Foto 4	7,57	6,22	4,30	6,57	8,67
Media	**7,75**	**6,46**	**4,60**	**7,32**	**9,49**
Imágenes generales					
Alemania	13,23	13,36	12,34	15,78	17,49
Lena	14,36	16,04	14,98	13,62	15,12
Loros	10,23	11,04	10,10	10,42	12,06
Pimientos	15,79	17,92	16,33	14,29	15,65
Media	**13,40**	**14,59**	**13,44**	**13,52**	**15,08**
Imágenes fractales					
Mand	9,00	10,39	7,72	9,70	11,27
Yinywthm	11,14	11,96	8,84	7,03	8,17
7thm	12,64	13,88	11,00	9,70	11,27
Atlantis	15,14	14,81	12,76	12,42	14,05
Media	**11,98**	**12,76**	**10,08**	**9,71**	**11,19**
Media Global	**11,04**	**11,27**	**9,38**	**10,35**	**12,19**

Tabla 7-13: Comparación de los algoritmos de crecimiento normal sin predicción, crecimiento normal sin predicción, y los estándares JPEG-LS y JBIG, además del algoritmo SLCIC

Podemos ver por primera vez, que en las imágenes médicas hay una diferencia sustancial en el valor de compresión colocando el bloque decorrelador después del crecimiento de regiones normal con vecindad 4. El valor de esa diferencia con respecto a JPEG-LS es de 0,86 bits/píxel mejorándolo. También supera en compresión al estándar JBIG para las imágenes médicas. Pero una vez más el algoritmo que alcanza unas cotas de compresión inmejorables para imágenes médicas, usando 1,86 bits/píxel menos, es el algoritmo SLCIC. En cuanto a las imágenes generales se observa un empeoramiento de la situación, así como también ocurre para las imágenes fractales.

En cuanto al número de regiones que generan uno y otro es el mismo porque lo único que cuenta para la generación de las regiones, es el crecimiento de regiones, y éste es el mismo en los dos esquemas, aquél que tiene el predictor y aquél que no lo tiene.

El coste computacional de introducir el decorrelador, frente al que no lo posee, es mínimo como se muestra en la Tabla 7-14.

Imágenes	*Crecimiento de vecindad 4 sin predictor (comp/descomp)*	*Crecimiento de vecindad 4 con predictor (comp/descomp)*
Fotografías de quemaduras		
Foto 1	4/1	4/1
Foto 2	4/1	4/1
Foto 3	4/1	4/1
Foto 4	4/1	4/1
Imágenes generales		
Alemania	3/1	3/1
Lena	2/1	2/1
Loros	3/1	3/1
Pimientos	2/1	2/1
Imágenes fractales		
Mand	1/1	1/1
Yinywthm	46/43	46/43
7thm	2/1	2/1
Atlantis	0/0	0/0

Tabla 7-14: Comparación de tiempos entre el esquema que posee el decorrelador y aquél que no lo posee. Tiempos medidos en segundo

Una vez obtenido este resultado en el que el decorrelador consigue buenos resultados en las imágenes médicas, el siguiente paso que vamos a proponer manteniendo el decorrelador o predictor, es suprimir el crecimiento de regiones a ver cuál es el resultado. Esto se hace porque se piensa que tenemos el crecimiento de regiones para eliminar la redundancia interpíxel, pero también usamos el predictor para dicho motivo, y queremos ver el resultado de probar un solo reductor de redundancia interpíxel por si no se complementaran bien ambos dos. El siguiente estudio de compresión estará compuesto por el esquema formado por el crecimiento normal de vecindad 4 y codificación aritmética, por el esquema con crecimiento normal, decorrelación y codificación aritmética, y el último esquema, el nuevo, con predictor y codificación aritmética. Por supuesto en este último caso no se necesitará codificar ninguna matriz de discontinuidades, ya que no hay crecimiento.

Imágenes	*Crecimiento normal de vecindad 4 y codificación aritmética sin predictor*	*Crecimiento normal de vecindad 4, predictor y codificación aritmética*	*Predictor y codificación aritmética*
Fotografías de quemaduras			
Foto 1	8,86	7,40	6,96
Foto 2	7,15	6,12	5,75
Foto 3	7,43	6,10	5,57
Foto 4	7,57	6,22	5,67
Media	**7,75**	**6,46**	**5,99**
Imágenes generales			
Alemania	13,23	13,36	11,92
Lena	14,36	16,04	14,41
Loros	10,23	11,04	9,85
Pimientos	15,79	17,92	15,89
Media	**13,40**	**14,59**	**13,02**
Imágenes fractales			
Mand	9,00	10,39	8,92
Yinywthm	11,14	11,96	9,16
7thm	12,64	13,88	11,43
Atlantis	15,14	14,81	13,56
Media	**11,98**	**12,76**	**10,77**
Media Global	**11,04**	**11,27**	**9,93**

Tabla 7-15: Comparación de tres esquemas de compresión. 1º crecimiento normal con vecindad 4 sin predictor y con codificación aritmética.2º Igual que el primero pero con un predictor después del crecimiento de regiones. 3º Predictor y codificación aritmética.

Los resultados que hemos obtenido en la Tabla 7-15 al suprimir la segmentación han sido realmente buenos en cuanto a las imágenes médicas, reduciendo al crecimiento de regiones sin predictor en 1,76 bits/píxel. En todas las imágenes de quemados, mejora a los dos anteriores. En cuanto a las imágenes generales, observamos también un buen resultado, ya que en media y analizando imagen a imagen, excepto una, supera a todos los valores de los otros dos esquemas. Las imágenes fractales proporcionan los mismos resultados que los otros dos grupos de imágenes. La diferencia en las fractales llega en media con respecto al crecimiento de regiones sin predictor y con predictor a 1,21 bits/píxel y 1,99 bits/píxel respectivamente. En media global tenemos unas diferencias de 1,11 bits/píxel y 1,34 bits/píxel, que son valores muy importantes.

El coste computacional ahorrado al suprimir el crecimiento de regiones se reparte de forma desigual como vemos en la Tabla 7-16.

Imágenes	*Crecimiento de vecindad 4 sin predictor (comp/descomp)*	*Predictor y codificación aritmética (comp/descomp)*
Fotografías de quemaduras		
Foto 1	4/1	4/1
Foto 2	4/1	3/1
Foto 3	4/1	3/1
Foto 4	4/1	4/1
Imágenes generales		
Alemania	3/1	3/1
Lena	2/1	2/0
Loros	3/1	3/1
Pimientos	2/1	2/0
Imágenes fractales		
Mand	1/1	0/0
Yinywthm	46/43	0/0
7thm	2/1	0/0
Atlantis	0/0	0/0

Tabla 7-16: Comparación de tiempos entre el esquema que posee crecimiento de regiones de vecindad 4 y predictor, con el que posee sólo decorrelador. Tiempos medidos en segundos

Por ejemplo para las imágenes de quemados la reducción solo llega a un segundo en algunos casos y en las imágenes generales mejora la descompresión. Donde se ve una diferencia notable es en la reducción del tiempo de compresión y descompresión de *Yinywthm*.

Puesto que la supresión del crecimiento de regiones ha reportado una mejor tasa de compresión y también se han mejorado los tiempos de compresión/descompresión, nos vemos en la necesidad de seguir probando esquemas suprimiendo el crecimiento de regiones. Con esto hemos visto que, a pesar de que la segmentación funciona muy bien con el codificador JBIG, no lo hace tan bien con el codificador aritmético usado, y por lo tanto si queremos seguir usando el codificador aritmético y obtener mejores tasas de compresión, deberíamos suprimir el crecimiento de regiones.

Una vez que no vamos a usar el crecimiento de regiones, el esquema de compresión queda como: una transformación de espacio de color $Y_sI_sQ_s$, un predictor y un codificador aritmético. A continuación vamos a ver que lo que podemos modificar ahora es el codificador aritmético para que tenga una ganancia de compresión mayor.

7.5.4 Codificador aritmético con contexto

Como indica el título de este capítulo, es el turno de modificar el codificador aritmético. Después de haber suprimido el crecimiento de regiones, vamos a modificar el codificador aritmético. La mejora que se le puede introducir al codificador aritmético es añadirle contextos, es decir, que usemos un codificador aritmético en el que el modelo de probabilidad se inicialice a cero cada vez que vayamos a comprimir un contexto diferente. Con esta separación en contexto, lo que se intenta es reducir la desviación de los datos que entran en el codificador para cada contexto. Estamos obteniendo unas tasas de compresión bastante altas, que ya se igualan con el estándar JPEG-LS, pero la inclusión del contexto mejorará la compresión.

Hemos usado dos tipos de estimadores del contexto, uno en el que la estimación de dicho contexto se hacía promediando los datos de alrededor, y otro un poco más avanzado en el que el contexto se determinaba por los gradientes locales de la imagen que recogían la actividad de la imagen en una determinada zona. Al primero lo llamaremos normal y al segundo modificado.

Imágenes	*Predictor y codificación aritmética*	*Predictor y codificación aritmética con contexto normal*	*Predictor y codificación aritmética con contexto modificado*
Fotografías de quemaduras			
Foto 1	6,96	6,83	6,77
Foto 2	5,75	5,49	5,39
Foto 3	5,57	5,46	5,39
Foto 4	5,67	5,54	5,47
Media	**5,99**	**5,83**	**5,75**
Imágenes generales			
Alemania	11,92	11,80	11,83
Lena	14,41	14,24	14,24
Loros	9,85	9,60	9,62
Pimientos	15,89	15,78	15,74
Media	**13,02**	**12,86**	**12,85**
Imágenes fractales			
Mand	8,92	8,67	8,44
Yinywthm	9,16	7,87	7,47
7thm	11,43	10,35	10,02
Atlantis	13,56	12,38	12,10
Media	**10,77**	**9,82**	**9,50**
Media Global	**9,93**	**9,50**	**9,37**

Tabla 7-17: Comparación de tres esquemas de compresión. 1º codificación aritmética normal. 2º codificación aritmética con contexto normal. 3º codificación aritmética con contexto modificado.

Como podemos observa en la Tabla 7-17 los valores de las tasas de compresión son muy buenos. Comparando el codificador aritmético sin contexto, con los dos que poseen contexto, sea uno u otro, existe una mejora clara de compresión en media de 0,43 bits/píxel y 0,56 bits/píxel, según sea contexto normal o modificado. Exceptuando en las imágenes generales donde el codificador con contexto normal es 0,01 bits/píxel superior al que no tiene contexto, en todas las demás el contexto presenta mejoras más claras. Por lo tanto el esquema que elegiremos será uno con contexto. En cuanto a la diferencia entre el codificador aritmético con contexto normal y el de contexto modificado, se puede observar que en media global el contexto modificado mejora en 0,13bits/píxel al contexto normal. Viéndolo de grupo en grupo, en las imágenes médicas observamos una mejora de la compresión cuando usamos contexto modificado, de igual modo ocurre para las imágenes generales y las fractales. Esto es debido a que el contexto modificado usa gradientes locales para predecir el contexto al que pertenece un píxel, con lo que este método será mejor que un promediado.

Los tiempos que se emplean a la hora de comprimir la imagen, usando un codificado aritmético sin contexto y los dos con contextos son prácticamente los mismos, por no decir los mismos como vemos en la Tabla 7-18.

Imágenes	*Predictor y codificación aritmética (comp/decom)*	*Predictor y codificación aritmética con contexto normal (comp/decom)*	*Predictor y codificación aritmética con contexto modifi. (comp/decom)*
Fotografías de quemaduras			
Foto 1	4/1	4/1	4/1
Foto 2	3/1	3/1	3/1
Foto 3	3/1	3/1	3/1
Foto 4	4/1	4/1	4/1
Imágenes generales			
Alemania	3/1	3/1	3/1
Lena	2/0	2/0	2/0
Loros	3/1	3/1	3/1
Pimientos	2/0	2/0	2/0
Imágenes fractales			
Mand	0/0	0/0	0/0
Yinywthm	0/0	0/0	0/0
7thm	0/0	0/0	0/0
Atlantis	0/0	0/0	0/0

Tabla 7-18: Tiempos de compresión/descompresión para los esquemas de compresión arriba expuestos. El tiempo viene dado en segundos.

Sólo por curiosidad, después de haber efectuado estas mediciones, calculamos la compresión que se alcanzaría con un crecimiento normal de regiones de vecindad 4 con un codificador aritmético basado en el contexto, y su correspondiente sin crecimiento de regiones (Tabla 7-19).

Imágenes	*Crecimiento de vecindad 4 con predictor y cod. aritmética con contexto normal*	*Predictor y codificación aritmética con contexto normal*
Fotografías de quemaduras		
Foto 1	7,20	6,83
Foto 2	5,80	5,49
Foto 3	5,87	5,46
Foto 4	5,86	5,54
Media	**6,18**	**5,83**
Imágenes generales		
Alemania	13,48	11,80
Lena	16,04	14,24
Loros	11,04	9,60
Pimientos	17,93	15,78
Media	**14,62**	**12,86**
Imágenes fractales		
Mand	10,38	8,67
Yinywthm	10,15	7,87
7thm	12,76	10,35
Atlantis	13,68	12,38
Media	**11,74**	**9,82**
Media Global	**10,85**	**9,50**

Tabla 7-19: Comparación del codificador aritmético sobre un crecimiento de regiones normal con vecindad 4 y un predictor, y con el predictor sólo. Medidas realizadas en bits/píxel.

Observamos cómo aplicando el mismo codificador entrópico el crecimiento de regiones da resultados menores en cuanto a compresión. Esa fue la razón por la cual se optó por suprimir el crecimiento, porque parece que no funciona demasiado bien con el codificador aritmético, pero eso sí, el crecimiento y el codificador JBIG son un esquema de compresión de características muy buenas.

Con esto hemos llegado a un punto en el que podemos comparar los resultados que hemos conseguido, con los estándares JPEG-LS, JBIG y con el algoritmo SLCIC. El algoritmo que usaremos en la comparación es el que mejores resultados ha proporcionado, y es aquél, cuyo esquema de compresión comprende de transformación de espacio de color $Y_sI_sQ_s$, predictor y codificador aritmético con contexto modificado. Los resultados finales están en la Tabla 7-20.

Imágenes	*Predictor y codificación aritmética con contexto modificado*	*SLCIC*	*JPEG-LS*	*JBIG*
Fotografías de quemaduras				
Foto 1	6,77	5,85	9,65	12,43
Foto 2	5,39	4,02	6,37	8,19
Foto 3	5,39	4,21	6,68	8,68
Foto 4	5,47	4,30	6,57	8,67
Media	**5,75**	**4,60**	**7,32**	**9,49**
Imágenes generales				
Alemania	11,83	12,34	15,78	17,49
Lena	14,24	14,98	13,62	15,12
Loros	9,62	10,10	10,42	12,06
Pimientos	15,74	16,33	14,29	15,65
Media	**12,85**	**13,44**	**13,52**	**15,08**
Imágenes fractales				
Mand	8,44	7,72	9,70	11,27
Yinywthm	7,47	8,84	7,03	8,17
7thm	10,02	11,00	9,70	11,27
Atlantis	12,10	12,76	12,42	14,05
Media	**9,50**	**10,08**	**9,71**	**11,19**
Media Global	**9,37**	**9,38**	**10,35**	**12,19**

Tabla 7-20: Comparación del algoritmo desarrollado y los estándares JPEG-LS y JBIG, además del algoritmo SLCIC.

Como podemos observar en la Tabla 7-20, los resultados obtenidos son realmente buenos. Comparándolo con el algoritmo SLCIC obtenemos una compresión global prácticamente igual, sólo 0,01 bits/píxel mejora el de contexto modificado al SLCIC. La capacidad de compresión del algoritmo SLCIC en imágenes de quemados es muy alta . En las imágenes generales la compresión de nuestro algoritmo en media obtiene 0,59 bits/píxel menos que el algoritmo SLCIC. En cuanto a las imágenes fractales en media nuestro algoritmo usa 0,58 bits/píxel menos para almacenar la información.

En cuanto a la comparación con el algoritmo JBIG en todos los casos nuestro algoritmo lo mejora, habiendo una diferencia en cada uno de los grupos de: imágenes de quemados 3,74 bits/píxel, imágenes generales 2,23 bits/píxel, imágenes fractales 1,69 bits/píxel.

En cuanto a la comparación con el más conocido estándar de compresión sin pérdidas de la actualidad, el JPEG-LS, obtenemos de nuevo unos resultados muy buenos. En media global de los tres grupos de imágenes hemos obtenido una mejora con

nuestro algoritmo de 0,97 bits/píxel que aunque parezca una cifra baja, ya dijimos que en una imagen de una resolución de 832×624 puede suponer un ahorro en datos de unos 63Kbytes, lo cual es una cifra importante. En cuanto a los diferentes grupos de imágenes nuestro algoritmo mejora en compresión al JPEG_LS en los siguientes valores: imágenes médicas 1,57bits/píxel, imágenes generales 0,67 bits/píxel, imágenes fractales 0,21 bits/píxel. Estas mejoras se dan sin aumentar mucho el coste computacional.

CAPÍTULO 8

CONCLUSIONES Y LÍNEAS FUTURAS

El objetivo de este proyecto ha sido ir desarrollando diferentes algoritmos de compresión a partir de uno, e ir viendo como la inclusión de un bloque mejoraba o empeoraba la eficiencia del esquema, hasta llegar a uno que mejorase al estándar JPEG-LS.

Como paso inicial y como primera aportación se comparó el algoritmo SLCIC que posee crecimiento de regiones de vecindad 4 con el algoritmo desarrollado que poseía vecindad 8 y con los demás mismos bloques que el SLCIC, comprobando que se obtenían mejores resultados en compresión para vecindad 4 que para 8. A partir de este momento todos los algoritmos posteriores con segmentación usarían crecimiento de vecindad 4.

En el siguiente paso, se incluyó un codificador entrópico diferente del que partíamos con SLCIC. En SLCIC teníamos un codificador JBIG, ahora lo cambiamos por un codificador aritmético. Éste es un paso intermedio de los que vienen después, ya que el codificador JBIG además de poseer un codificador entrópico, poseía un decorrelador previo que hacía que sus resultados fueran mejores que usando el aritmético.

Por lo tanto se veía la necesidad de mejorar el conjunto de datos que se le pasaban al codificador. Para ello se modificó el crecimiento de regiones haciendo que el número de bits de error fuera variable, con lo que el rango dinámico de los datos sería menor. Los resultados de esta modificación no fueron los esperados, ya que empeoraba este tipo de crecimiento al normal de vecindad 4, por lo que este tipo de crecimiento se desecho.

Una vez visto que el crecimiento de regiones modificado no funcionaba muy bien con el codificador aritmético, se necesitaba otro tipo de bloque que redujera la

correlación entre los datos que se le pasaban al codificador. Se pensó y se uso el predictor o decorrelador del estándar JPEG-LS. Éste se probó antes y después del crecimiento de regiones llegando a la conclusión que previo al crecimiento de regiones no era viable. Los resultados obtenidos con esta inclusión mejoraron los anteriores, por lo que se convirtió en un bloque del esquema.

Viendo que el crecimiento de regiones no funcionaba muy bien con el codificador aritmético, el siguiente paso fue quitar el crecimiento de regiones y dejar sólo el predictor. Los resultados obtenidos con este esquema fueron mejores que los que poseían segmentación, por lo que se pensó en suprimir la segmentación.

El último paso fue mejorar el codificador aritmético que estábamos usando. Se le incluyeron dos tipos de contexto, uno llamado normal y otro llamado modificado. El primero estaba basado en la predicción por ponderación y el segundo por gradientes locales. Los resultados obtenidos por uno y otro mejoraban el codificador aritmético normal y el de gradientes locales mejoraba la compresión frente al de ponderación.

En la Figura 8-1 se muestra la línea que hemos seguido a lo largo de todo el desarrollo de este proyecto, y de cómo hemos ido desechando alternativas dependiendo de los resultados obtenidos. Los círculos en gris son decidores que determinan cuál de los esquemas puestos en prueba obtiene mejores resultados. Ésta ha sido la línea seguida hasta llegar a un algoritmo de compresión de imágenes basado en una transformación de espacio de color $Y_sI_sQ_s$, un predictor y un codificador aritmético con contexto determinado por gradientes locales. Los resultados obtenidos por este compresor son muy buenos teniendo en cuenta que mejora el más conocido estándar de compresión sin pérdidas, el algoritmo JPEG-LS.

Descrita la línea seguida en el desarrollo de este proyecto se exponen las siguientes conclusiones:

- La segmentación de la imagen con vecindad 4 mejora la tasa de compresión frente a la de vecindad 8.

- El crecimiento de regiones con codificación aritmética no mejora la compresión frente al codificador JBIG.

- El uso de crecimiento de regiones con número de bits de error variable no es viable debido a la sobrecarga de la matriz de discontinuidad.

- Con codificación aritmética, el proceso de segmentación empeora frente a aquél que no lo usa, como es el caso del predictor solo.

- Se consigue una mejora considerable introduciendo contextos en el codificador aritmético, ya sean de ponderación o por medio de gradientes locales, superando éstos últimos a los primeros.

Viendo la línea que se ha seguido, se ha llegado a un algoritmo de compresión de imágenes sin pérdidas que mejora al estándar JPEG-LS en casi 1bit/píxel de media y sin apreciarse un alto incremento de coste computacional. En cuanto al algoritmo SLCIC prácticamente se igualan las tasas de compresión mejorando sólo en 0,01bits/píxel al SLCIC.

Como futuras líneas de investigación se proponen:

- Intentar mejorar el predictor de la imagen para generar la imagen error. Se podría usar un mayor número de píxeles en el entorno a predecir como hace el algoritmo CALIC.

- Cambiar por completo el esquema de inclusión de píxeles en las regiones en el crecimiento de regiones, visto que éste no obtiene buenos resultados con el codificador aritmético.

- En cuanto al contexto del codificador aritmético, intentar mejorar la predicción del contexto con vistas a aumentar el número de contextos. Existe un compromiso entre tasas de compresión y coste computacional. Conforme aumentemos el número de contextos, la compresión mejorará, pero el coste computacional aumentará. Para decir que la compresión mejora, se debe cumplir que la predicción de un píxel dentro de un contexto o de otro sea fiable. Para aumentar la fiabilidad de predecir el contexto habrá que mejorar el predictor de contexto.

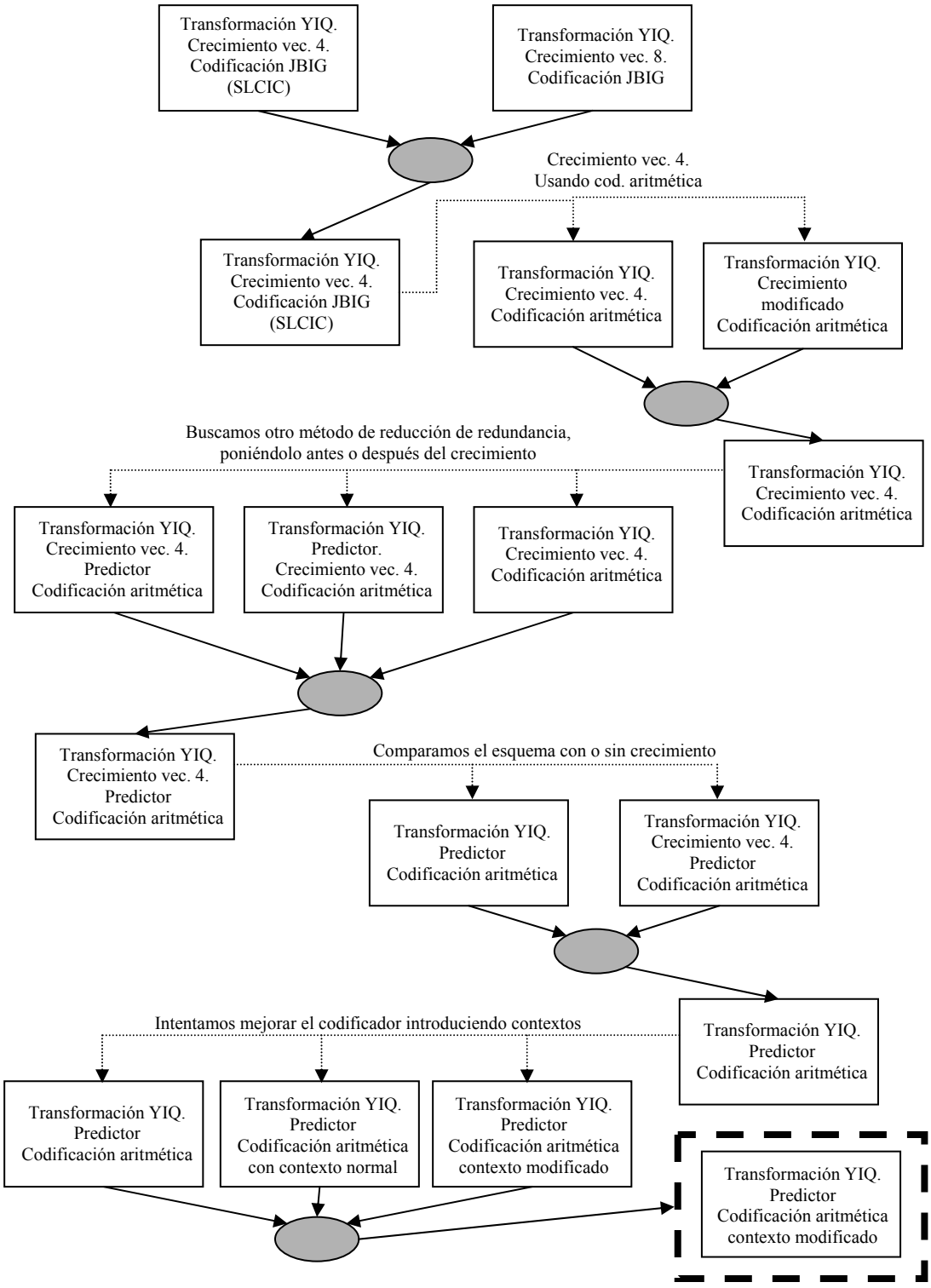


Figura 8-1 : Línea seguida en los desarrollos de los algoritmos de compresión sin pérdidas.

APÉNDICE A

FORMATO BMP

A.1 INTRODUCCIÓN

En este apéndice se describe el formato de mapa de bits para imágenes de Microsoft Windows y IBM OS/2, que se denominan *Bitmap* o archivos BMP. La mayoría de la descripción del archivo BMP se concentra en las estructuras `BMPINFOHEADER` y `BMPCOREINFO`, las cuales son necesarias si queremos trabajar leyendo y escribiendo imágenes BMP.

A.2 FORMATO DE ARCHIVO BITMAP

El formato BMP fue creado por Microsoft e IBM y es muy estricto en cuanto a arquitectura de la plataforma hardware que estas dos compañías soportan, esto es, los famosos PC. Esto significa que los valores almacenados en el formato BMP están en formato Intel, también llamado *Little Endian.*

Cada línea de la imagen BMP debe terminar en un *dword*, y si no ocurre así se deben añadir ceros.

El formato BMP tiene cuatro versiones, dos bajo Windows y dos bajo OS/2, las cuales se describen todas aquí. La siguiente tabla contiene una descripción del archivo BMP.

	Campo	Tamaño	Descripción
HEADER. Windows Structure: BITMAPFILEHEADER (14 bytes)	Identificador	2 bytes	Los caracteres identifican el Bitmap. Estos son los valores posibles: 'BM' - Windows 3.1x, 95,98,ME,XP, NT, … 'BA' - OS/2 Bitmap Array 'CI' - OS/2 Color Icon 'CP' - OS/2 Color Pointer 'IC' - OS/2 Icon 'PT' - OS/2 Pointer
	Tamaño de archivo	4 bytes	Tamaño completo del archivo en bytes.
	Reservado	4 bytes	Reservado para uso posterior.
	Offset de los datos del Bitmap	4 bytes	Offset desde el comienzo del archivo hasta el comienzo de los datos del bitmap.
INFOHEADER. Windows Structure: BITMAPINFOHEADER (40 bytes)	Tamaño del Header del Bitmap	4 bytes	Longitud de Bitmap Info Header usado para describir los colores del bitmap, compresión... Los tamaños posibles son: 28h - Windows 3.1x, 95, NT, … 0Ch - OS/2 1.x F0h - OS/2 2.x
	Ancho	4 bytes	Anchura de la imagen en pìxeles.
	Alto	4 bytes	Alto de la imagen píxeles.
	Planos	2 bytes	Numero de planos (=1)
	Bits por Pixel	2 bytes	Bits por píxel usados para almacenar la información de la tabla de colores. Con esto es posible saber el número de posibles colores. Los valores posibles son: 1 - Bitmap monocromo 4 - Bitmap 16 color 8 - Bitmap 256 color 16 - Bitmap 16bit (high color) 24 - Bitmap 24bit (color verdadero) 32 - Bitmap 32bit (color verdadero)
	Compresión	4 bytes	Tipo de compresión. Los siguientes valores son validos: 0 - BI_RGB No hay compresión 1 - RLE 8-bit / píxel (También BI_RLE8) 2 - RLE 4-bit / píxel (También BI_RLE4) 3 - Bitfields (También BI_BITFIELDS)
	Tamaño de la imagen	4 bytes	Tamaño de la imagen en bytes. Es válido poner este campo a 0 si no hay compresión.
	HResolucion	4 bytes	Resolución horizontal expresada en píxel por metro.
	VResolution	4 bytes	Resolución vertical expresada en píxel por metro.
	Colores usados	4 bytes	Número de colores usado es en el bitmap.
	Colores importantes	4 bytes	Número de colores importantes. Este número será igual al número de colores cuando cada color sea importante.

TABLA DE COLORES	Paleta o Tabla de colores	N * 4 byte	Especificación de la paleta. Para cada entrada en la paleta hay cuatro bytes para describir los valores RGB de la siguiente forma: 1 byte para la componente B(Blue) 1 byte para la componente G(Green) 1 byte para la componente R(Red) 1 byte de relleno con valor 0(cero)
PÍXEL DATA	Datos del bitmap	Info. Image Size(bytes)	Los datos de los píxeles.

Tabla A-1: Esquema general de un archivo BMP

A continuación se describen algunos campos que requieren más información.

A.2.1 Bits por píxel

El número de bits por píxel determina el número de bits que definen cada píxel y el máximo número de colores en el bitmap.

- **Bits por píxel = 1**

Este bitmap es monocromo, y la paleta contiene dos entradas. Cada bit en la tabla bitmap representa a un píxel. Si el bit es *clear*, el píxel posee el valor de la primera entrada en la paleta. Por el contrario si el bit está *set*, el píxel se corresponde con la segunda entrada en la tabla

- **Bits por píxel = 4**

El bitmap tiene un máximo de 16 colores, y la paleta puede contener hasta un máximo de 16 colores. Cada píxel está representado por un índice de 4 bits que accede a la paleta de colores. Si el primer byte en el bitmap es 1Fh, ese byte representa a dos píxeles.

- **Bits por píxel = 8**

Este bitmap tiene un máximo de 256 colores, y la paleta puede albergar hasta 256 entradas de colores. En este caso cada byte de la tabla de bitmap representa un píxel de la imagen.

- **Bits por píxel = 16**

El bitmap tiene un máximo de 2^{16} colores. Cada word en la tabla de bitmap representa a un píxel. Las intensidades de rojo, verde y azul son representadas con 5 bits para cada color. El valor del azul está en los 5 LSBs, es seguido por los bits del color verde y el rojo respectivamente. El MSB no se usa.

- **Bits por píxel = 24**

El bitmap tiene un máximo de 2^{24} colores, y la paleta no contiene ninguna entrada. Cada triada de bytes se corresponde con un píxel, representando cada byte, cada una de las intensidades de azul, verde y rojo.

A.2.2 Campo de compresión

Este campo especifica la forma en que la información del bitmap es almacenada. Esta información junto con el número de bits por píxel identifica el algoritmo de compresión a seguir. Los valores posibles en este campo son los siguientes

Valor	***Significado***
BI_RGB	Corresponde al formato sin compresión
BI_RLE4	Formato RLE (run length encoded) para bitmaps de 4 bits/píxel. El formato de compresión es uno de 2-bytes que cuenta los bytes seguidos de índices de longitud 2 word.
BI_RLE8	Formato RLE (run length encoded) para bitmaps de 8 bits/píxel. Consiste en un formato de 2 bytes seguidos de un byte que indica el índice del color.
BI_BITFIELDS	Especifica que el bitmap no tiene compresión y que la tabal de colores consiste en 3 máscaras de color de double word que especifican las componentes roja, verde y azul, respectivamente, de cada píxel. Válido cuando se usa 16 y 32 bits por píxel.

Si el campo de compresión tiene el valor BI_RLE8, el bitmap está comprimido usando un formato *run length encoding* (RLE). Este formato puede ser comprimido de dos modos, a saber, *encoded* o *absolute*.

- **Modo *encoded* formado por 2 bytes:**

El primer byte especifica el número de píxeles consecutivos que siguen y que tienen como índice de color el contenido en el segundo byte. Además, el primer byte puede ser 0 para indicar fin de línea, fin de bitmap o delta. La interpretación depende del valor del segundo byte, que puede ser uno de los siguientes:

0	Fin de línea
1	Fin de bitmap
2	Delta. Los dos bytes siguientes contienen valores sin signo que indican la posición que ha de desplazarse horizontal y verticalmente para llegar al siguiente píxel, viniendo del actual.

- **Modo *absolute***

El primer byte es 0 y el segundo byte es un valor en el rango de 03h a FFh. El segundo byte representa el número de bytes que siguen, cada uno de los cuales contiene el índice de color. Cuando el segundo byte es 2 o menos, tiene el mismo significado el que sigue que en el modo *encoded* tenía el segundo byte cuando el primero estaba puesto a 0.

Cuando el valor de compresión es BI_RLE4, el bitmap está comprimido usando un run-lenth encoding para 4 bits. Igual que en el caso de 8 bits puede usarse en dos modos *encoded* y a*bsolute*,

- **Modo *encoded***

El primer byte contiene el número de píxeles que se pondrán usando los índices de color del segundo byte. El segundo byte contiene dos índices de color uno llamado high-order y otro llamado low-order, los píxeles se pintarán, el primero usara el high-order, el siguiente usará el low-order, el tercero usará el high-order y así sucesivamente.

- **Modo a*bsolute***

El primer byte es 0, el segundo contiene el número índices de color que siguen, y los siguientes bytes usarán el high-order y el low-order cada uno de 4 bits para cada píxel.

El fin de línea, el fin de bitmap y la delta tienen el mismo funcionamiento que en el BI_RLE8.

REFERENCIAS

[Castleman, 96] K.R. Castleman. *Digital Image Processing.* Prentice Hall, Englewood Cliffs,New Jersey 1996.

[González, 96] R.C. González, R.E. Woods. *Tratamiento digital de imágenes*. Addison-Wesley Iberoamericana, Wilmington 1996.

[Haykin, 88] S. Haykin. Digital Communications. John Wiley & Sons, New York 1988.

[Jain, 89] A.K. Jain, *Fundamental of Digital Image Processing*, Prentice Hall, 1989.

[Lim, 90] J.S. Lim, Two-*dimensial signal and image processing*. Prentice Hall, New Jersey 1990.

[Merhav, 97] N. Merhav, G. Seroussi, M. J. Weinberger, "Coding of Sources with Two-Sided Geometric Distributions and Unknown Parameters". IEEE Transactions on Information Theory, Vol. 46, No. 1, January 2000.

[Proakis, 95] J.G. Proakis. *Digital Communications*. McGraw-Hill, Malaysia 1995.

[Proakis, 98] J.G. Proakis, D.G. Manolakis. *Tratamiento Digital de Señales*. Prentice Hall, Madrid 1998.

[Robinson, 97] J. A. Robinson, "Efficient General-Purpose Image Compression with Binary Tree Predictive Coding". *IEEE Trans. Image Processing,* Vol. 6, April 1997 pp. 601-608.

[Serrano, 99] C. Serrano, B. Acha, R.M. Rangayyan, "Segmentation-Based Lossless Compression for Color Images", *Proc. Int. Conf. On Image Analysis and Processing*, pp. 90-94, Venecia 1999.

[Serrano, 01] C. Serrano, B. Acha, R.M. Rangayyan, L.M. Roa, "Segmentation-based lossless compression of burn wound images", *Journal of Electronic Imaging* 10(3): 720-726, Julio 2001.

[Serrano, 01] C. Serrano. "*Algoritmo de Compresión sin Pérdidas de Imágenes a Color basado en Segmentación*". Universidad de Sevilla. Sevilla, 2001.

[Singh, 97] I. Singh, P. Agathoklis, A. Antoniou, "Compression of color images using mixed transform techniques". *Univesrity of Victoria*. 1997.

[Triantafyllidis, 99] G.A. Triantafyllidis, M.G. Strintzis, "A context based adaptative coding technique for lossless image compression". *IEEE Signal Processing Letters* Vol 6 No 7 July 1999.

[UIT-T, 93] Unión Internacional de Telecomucnicaciones, " Progressive bi-level image compression", *Recomendación T.82*, 1993.

[Wallace, 82] G.K. Wallace, "The JPEG still picture compression standard". *IEEE Trans. on Consumer Electronics,* Vol. 38, No 1 February 1982.

[Weinberger, 99] M. Weinberger, G. Seroussi and G. Sapiro, "From LOCO-I to JPEG-LS standar". *IEEE Trans. Image Processing,* 1999, pp.68-72.

[Weinberger, 00] M. Weinberger, G. Seroussi and G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS". *IEEE Trans. Image Processing,* Vol. 9, August 2000, pp.1309-1324.

[Weinberger, 00] M.J. Weinberger, G. Seroussi, G. Sapiro "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS". *Hewlett-Packard Laboratories*, Palo Alto, CA 94304 2000

[Weinberger, 00] M.J. Weinberger, G. Seroussi, G. Sapiro "LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm". *Hewlett-Packard Laboratories*, Palo Alto, CA 94304 2000

[Witten, 87] I.H. Witten, R.M. Neal, J.G. Cleary, "Arithmetic coding for data compression", Communications of the ACM, Volume 30, Number 6, pp 520-540, June 1987.

[Wu, 96] X. Wu, N. Memon, "Calic, a context based adaptive lossless image code." *Proceedings of the data compression conference*, IEEE 1996 pp.1893.

[Wu, 00] X. Wu, N. Memon, "Context-based lossless interband compression-Extending Calic" *IEEE Transactoins on Image Processing*. Vol 6, No 6, June 2000