Máster en
Inteligencia Artificial y Big Data en Salud
A4. Adquisición, filtrado y seguridad de datos

Joan Bartrina, Bernat Gastón, Ramon Martí
Joan Oliver, Marta Prim, Mercè Villanueva

Institut Universitari Parc Taulí - UAB

Contents

Chapter 6

Compresión de Datos I

Joan Bartrina

A lo largo de esta unidad vamos a ver de forma superficial bastantes conceptos relacionados con la compresión de datos. Motivaremos su aplicación, introduciremos distintas unidades de almacenamiento, y veremos algunas definiciones sobre imágenes y vídeos. Además, se introducirán distintos términos relacionados con la eficiencia de compresión, aprenderemos a comparar dos imágenes (en términos de calidad) y se introducirán distintas técnicas de compresión sin y con pérdida.

6.1 Por qué la compresión de datos?

Actualmente los sistemas de adquisición de datos son más usados en el día a día de un centro médico. Des de la adquisición de datos en los procesos de generación de informes médicos, facturación y sistemas de gestión de citas, etc, hasta la recolección de imágenes médicas como radiografías, mamografías, tomografías computacionales, angiografias, etc. Uno de los principales problemas de los centros médicos es el de cómo almacenar toda esta información de forma adecuada.

De un modo muy general, el flujo de los datos en un sistema de compresión es el descrito por la figura 6.1. Los informes, archivos xml, facturas, albaranes, imágenes, y vídeos son comprimidos mediante un "compresor". Este compresor debe de ser específico para cada uno del tipo de datos a comprimir, ya que no existe un sistema de compresión genérico para cualquier tipo de datos (para más información se sugiere una lectura al Pigeon Theorem [1]). Un vez comprimidos los datos, estos se encontraran en un formato específico como: zip, rar, jpeg, jpg, jp2, avi, mpeg, mp4, mp5, hevc, etc. Para poder visualizar los datos correctamente deberemos descomprimirlos

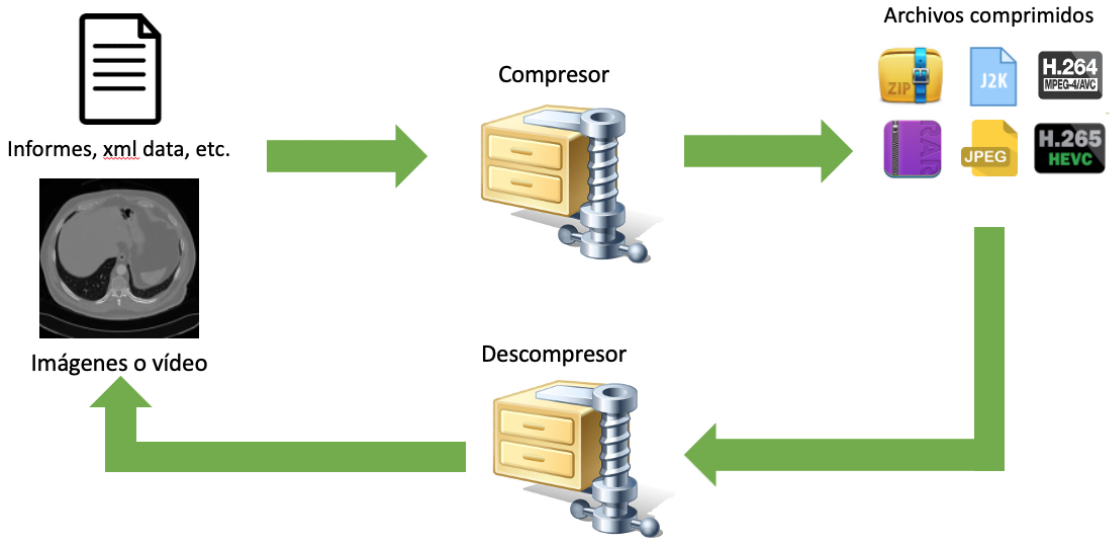


Figure 6.1: Flujo de datos.

previamente, motivo por el cual es esencial conocer el sistema de compresión utilizado, para poder descomprimir el archivo y poder visualizar los datos de forma correcta.

6.2 Unidades de almacenamiento en sistemas informáticos

En los sistemas informáticos los datos se almacenan utilizando bits, un bit es un 0 o 1. Una secuencia de 8 bits es 1 byte, 1024 bytes es un kilobyte (KB), 1024 KB son un megabyte (MB), 1024 MB son un gigabyte (GB), etc. En [2] se describen las distintas unidades de medida para sistemas informáticos.

Un secuencia de 0s y 1s nos permite almacenar y/o representar los datos deseados. Por ejemplo, en el caso de un archivo de texto debemos almacenar distintos caracteres: a, A, b, B, c, C, 1, 2, 3, 4, @, #, $, etc.

Para almacenar estos datos debemos utilizar una codificación específica, por ejemplo y para facilitar la comprensión del resto del documento utilizaremos la codificación ASCII [3]. Esta codificación almacena los datos que se van a imprimir por la pantalla, ya sean números o caracteres, en secuencias de 8 bits. La tabla 6.1 muestra para algunos caracteres, el valor decimal, y la secuencia de 0s y 1s, conocida como codificación binaria, que se almacena. Para los caracteres utilizaremos la codificación ASCII (Glyph 1967).

Carácter	Codificación decimal	Codificación binaria
@	64	0100 0001
A	65	0100 0001
C	67	0100 0011
P	80	0101 0000
S	83	0101 0011
0	48	0011 0000
1	49	0011 0001
.	.	.
.	.	.
.	.	.

Tabla 6.1: Algunos ejemplos de la relación entre caracteres, valor decimal y codificación binaria para ASCII (Glyph 1967).

Las ventajas de la utilización de la compresión de datos son bastante directas, ya que esta nos da la capacidad de almacenar en el mismo espacio (p.e Megabytes) más información, hecho que nos permite transmitir la misma información en un menor tiempo o utilizando menos ancho de banda (redes 3G, 4G o 5G). De aquí es importante destacar que lo relevante es como se almacena una misma información (p.e una secuencia de bits) de modo que esta se pueda representar con una secuencia binaria de menor longitud. Para ello, utilizamos una codificación. Es importante diferenciar entre datos e información. Por ejemplo, supongamos que alguien nos envía dos veces una mismo correo electrónico con una imagen médica que ocupa 500MB. Tenemos 2 veces la misma imagen, ocupando 1000MB de datos pero sólo 500MB nos aportan información.

Calcular lo que no puede ocupar un archivo de texto o una imagen dependerá de dos factores: 1) cantidad de caracteres para un archivo de texto o cantidad de píxeles o samples para una imagen. Y, 2) tipo de datos almacenados en un carácter o píxel. Vamos a ver unos ejemplos.

1. **Archivo de texto:** supongamos que tenemos un archivo de 800 páginas, cada página contiene 50 lineas y cada linea 60 caracteres. El número de caracteres del libro será de $800 \times 50 \times 60 = 2400000$. Si cada carácter se almacena utilizando la codificación ASCII necesitaremos 8 bits para cada carácter, por lo que le número de bits necesarios será de $2400000 \times 8 = 19200000$ bits. Podemos cambiar la medida de capacidad a megabyte (MB) para hacerlo más inteligible mediante

$$MB = \frac{bits}{8 * 1024 * 1024} \tag{6.1}$$

De modo que, el libro ocuparía un espacio en disco **sin comprimir** de 2.28 Mb.

2. **Imagen:** en este caso, en lugar de un archivo de texto disponemos de una tomografía computacional (CT). Las CTs se caracterizan por ser una secuencia de imágenes 2D obtenidas de una sección del cuerpo humano. Si cada una de las secuencias 2D ocupa 1024×1024 y esta formada por 220 imágenes 2D, el total de muestras (muestras) de la CT son $1024 \times 1024 \times 220 = 230686720$.

   Ahora nos falta por definir la cantidad de información almacenada en una muestra. En este caso, supongamos que el dispositivo adquiere datos que van de un rango comprendido entre el 0 y 1024. Para conocer cuantos bits necesitamos para almacenar cada muestra utilizaremos el concepto de entropía [4], concretamente la siguiente función la función:

   $$\log_2(\text{rango}) = \text{bits}. \tag{6.2}$$

   En donde el rango es la diferencia entre el valor máximo y mínimo adquirido por el dispositivo, en nuestro ejemplo máximo es 1024 y mínimo 0, de modo que necesitaremos $log_2(1024) = 10$ bits para almacenar cada muestra[1]. Como no podemos utilizar $8 + 2$ bits para almacenar una muestra, tendremos que guardar cada una de ellas en 2 bytes. En consecuencia, al final, la CT que hemos definido ocupa un total de

   $$\frac{230686720 * 2 * 8}{8 * 1024 * 1024} = 440 \text{ MB} \tag{6.3}$$

3. **Vídeo:** en los vídeos es importante tener en cuenta el concepto de "frame rate". El frame rate nos indica el nombre de imágenes que vemos por segundo, en inglés, frames per second (fps). Por ejemplo, si disponemos de una angiografia de 1024 filas y 1024 columnas con una profundidad de 12 bits por sample y un fps de 24 y de una duración de 5 minutos, esta angiografia sin comprimir ocupará

   $$\frac{1024 * 1024 * 2 * 8 * 24 * 5 * 60}{8 * 1024 * 1024 * 1024} = 14,06 \text{ GB} \tag{6.4}$$

Ahora bien, es importante utilizar las técnicas de compresión adecuadas en función de los datos a comprimir. Es muy diferente comprimir datos de texto, imágenes o vídeos.

[1]Este mismo concepto se aplica al almacenar texto, donde si tenemos que el valor máximo es 127 y el mínimo 0, necesitaremos $log_2(127) = 8$ bits para cada letra a almacenar.

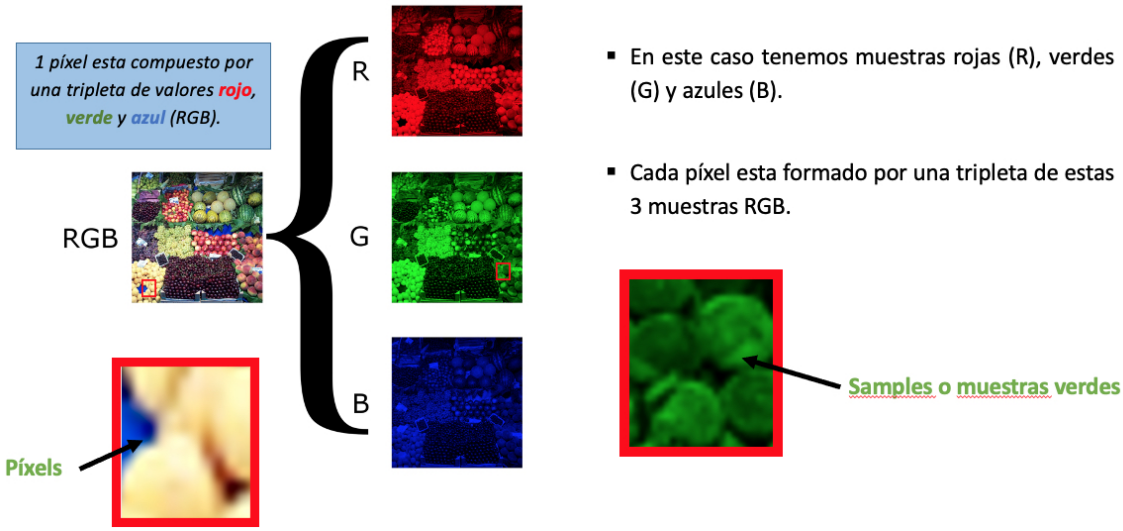


Figure 6.2: Diferencia entre píxel y sample.

6.2.1 Sample vs Píxel

A lo largo de esta sección hemos hablado de sample o muestra, pero habitualmente en imágenes digitales hablamos de píxeles. Es importante distinguir la diferencia entre estos dos términos. Mientras que un sample es una muestra única, un píxel es un conjunto de muestras de una misma posición espacial. Supongamos una imagen como un vector bidemensional X_{ijk}, donde i y j identifican las filas y columnas, respectivamente y k las componentes. Cada uno de los X_{ijk} se conoce como sample, mientras que el conjunto de valores del vector X_{ij} de longitud K se conoce como píxel. La figura 6.2 muestra un ejemplo visual de la diferencia entre píxel y sample.

6.3 Eficiencia de la compresión

Para medir la eficiencia de las distintas técnicas de compresión se utilizan diferentes métricas, las dos más comunes son el "Compression Ratio (CR)" y los "Bits per sample (bps)". El CR que se define como el ratio entre los datos sin comprimir y comprimidos, y es expresa mediante la siguiente fórmula:

$$CR = \frac{\text{Uncompressed Size}}{\text{Compressed size}}, \tag{6.5}$$

por otro lado los bps nos indican los bits necesarios para representar cada muestra, y se define como:

$$bps = \frac{\text{Compressed Size (in bits)}}{\text{Number of muestras to compress}}, \tag{6.6}$$

Habitualmente los sistemas de compresión **sin pérdida**, en inglés **lossless**, obtienen unas tasas de compresión de 3 a 1. Es decir necesitan 1 bit para cada 3 de la muestra original, de modo que su CR es aproximadamente 3. En algunos casos esta tasa de compresión no es suficiente para ello se recorre a utilizar técnicas de **compresión con pérdida**, en inglés **lossy**. Estas técnicas obtienen CR de 4, 5, 10, 100 o más, pero en contrapartida introducen cierta distorsión por lo que no se pueden recuperar los datos originales. Evidentemente a mayor CR, más distorsión y menos fiabilidad sobre los datos originales tendremos. Veamos dos ejemplos a partir de los datos definidos anteriormente.

1. **Compresión texto:** Tenemos que cada uno de los caracteres se almacena utilizando 8 bits. Si necesitamos almacenar la palabra "CASA", nuestra computadora almacenara "67,65,83,65" en codificación decimal y en codificación binaria tendremos las siguientes secuencias "01000011, 01000001, 01010011, 01000001", siguiendo la codificación de la tabla 6.1.

   Supongamos: 1) que hemos definido un sistema de compresión lossy, obtenemos CR $>$ 3, pero no somos capaces de recuperar los datos originales. Y, 2) que la pérdida introducida no nos permite recuperar los 2 últimos bits de cada carácter. De modo que, al descomprimir recuperaremos los siguientes datos:

   "010000XX,
   010000XX,
   010100XX,
   010000XX",

   donde la X son bits que no conocemos, de modo que los debemos "suponer". El hecho de desconocer estos bits NO nos permite recuperar los datos originales, por lo que introduciremos algún tipo de pérdida de información. En el caso de la compresión de texto, vamos a ver de forma muy simple que perder información no es asumible. Por ejemplo, si los datos desconocidos los recuperamos todos como 0 recuperaremos los siguientes datos:

   "01000000,
   01000000,
   01010000,
   01000000",

En decimal esto seria equivalente a “64, 64, 80, 64”. Utilizando la codificación ASCII [3] (ver tabla 6.1) obtenemos los siguientes caracteres “@, @, P, @”. Como se puede observar introducir pérdida en un archivo de texto recupera unos datos que son totalmente distintos a los originales, haciendo que las técnicas lossy no sean factibles para la compresión de texto.

2. **Compresión imágenes:** en este caso disponemos de la imagen CT descrita anteriormente que ocupa 440 Mb. Cada uno de las muestras o píxeles se encuentra almacenado en 16 bits, de modo que para representar un valor de muestro de 1323 almacenaremos la secuencia binaria “**0000** 0101 0010 1011”, donde los 4 primeros bits son conocidos como bits de relleno.

   La imagen original ocupa 16 bps, ya que por cada sample necesitamos 16 bits. La representación binaria de los valores decimales nos permite ir refinando el valor decimal a medida que conocemos más bits. Esto traducido en compresión es utilizado por las técnicas lossy para introducir pérdida degradando progresivamente la imagen.

   Supongamos: cómo en el ejemplo anterior que 1) hemos definido un sistema de compresión lossy, obtenemos CR > 3, pero no somos capaces de recuperar los datos originales. Y, 2) que la pérdida introducida no nos permite recuperar los N últimos bits de cada carácter. La tabla 6.2 muestra la relación entre los N bits que NO se conocen –debido a la pérdida introducida por el sistema de compresión–, la secuencia binaria obtenida, los bps necesarios para los datos, el valor decimal recuperado y el error que estamos cometiendo respecto al valor original de 1323. Es evidente que nivel analítico el hecho de introducir pérdida hace que los valores no se recuperen de forma correcta, pero a nivel visual que impacto tiene? hasta que punto podemos ir introduciendo pérdida?

   La Figura 6.3 (a) muestra una componente (slice) de una CT. Mientras que las figuras (b), (c) y (d) muestran la misma slice habiendo introducido una pérdida de 2, 6 y 8 bits. Se puede apreciar, que a nivel visual una pérdida de 2 e incluso 6 bits no produce ningún deterioro significativo a nivel visual. Ahora bien, con una pérdida de 8 bits si podemos apreciar pérdida de calidad visual –sobretodo si hacemos zoom–.

   Para medir el error introducido durante el proceso de compresión sobre los datos originales utilizaremos métricas de distorsión, concepto que abordaremos en la siguiente sección.

N	bps	Secuencia binaria	Valor decimal recuperado	Error
0	16	0000 0101 0010 1011	1323	1323 - 1323 = 0
1	15	0000 0101 0010 101**0**	1322	1323 - 1322 = 1
2	14	0000 0101 0010 10**00**	1320	1323 - 1320 = 3
3	13	0000 0101 0010 1**000**	1320	1323 - 1320 = 3
4	12	0000 0101 0010 **0000**	1312	1323 - 1312 = 11
5	11	0000 0101 001**0 0000**	1312	1323 - 1312 = 11
6	10	0000 0101 00**00 0000**	1280	1323 - 1280 = 43
7	9	0000 0101 0**000 0000**	1280	1323 - 1280 = 43
8	8	0000 0101 **0000 0000**	1280	1323 - 1280 = 43
9	7	0000 010**0 0000 0000**	1024	1323 - 1024 = 299
10	6	0000 01**00 0000 0000**	1024	1323 - 1024 = 299
11	5	0000 0**000 0000 0000**	0	1323 - 0 = 11

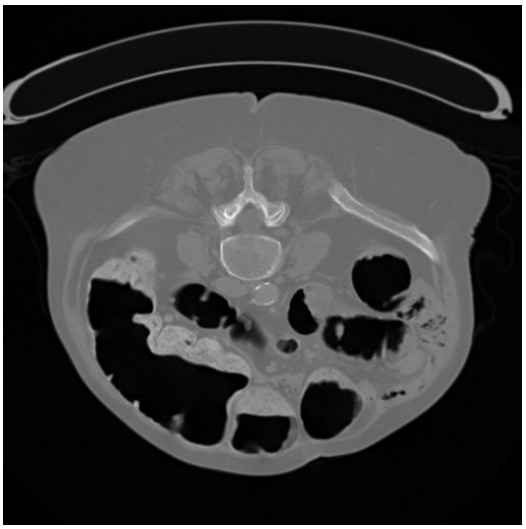
Tabla 6.2: Ejemplo de pérdida de información según la cantidad de bits desconocida debido a la compresión lossy.

> *El resto de la materia de las unidades de compresión se centraran, principalmente, en la compresión de imágenes y/o vídeos. Esto viene motivado ya que los datos generados por los sistemas sanitarios son mayoritariamente imágenes o vídeos, siendo estos el gran problema a solucionar en cuanto a un almacenamiento y transmisión por la red lo más eficiente posible.*

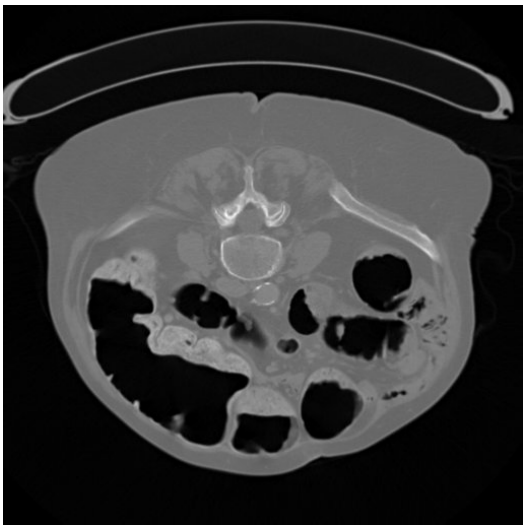
6.4 Principios básicos para la compresión

El objetivo de la compresión es la de mostrar la misma información utilizando una codificación que a su vez emplee menos símbolos y en consecuencia menos espacio que la original. De modo que la codificación convertirá un carácter de un lenguaje natural en un símbolo de otro sistema de representación.

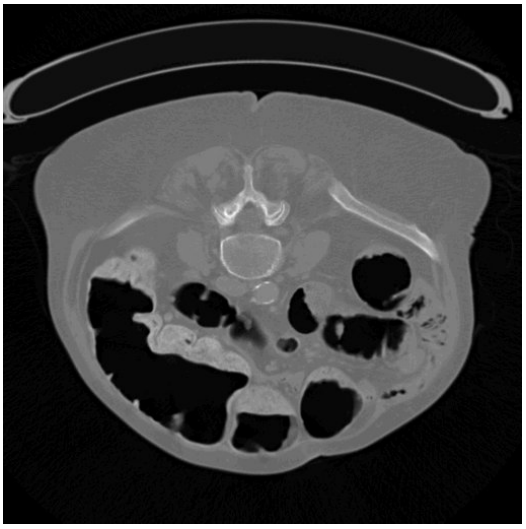
La información contenida en un mensaje es proporcional a la cantidad de bits que se requieren como mínimo para representar al mensaje. Para medir



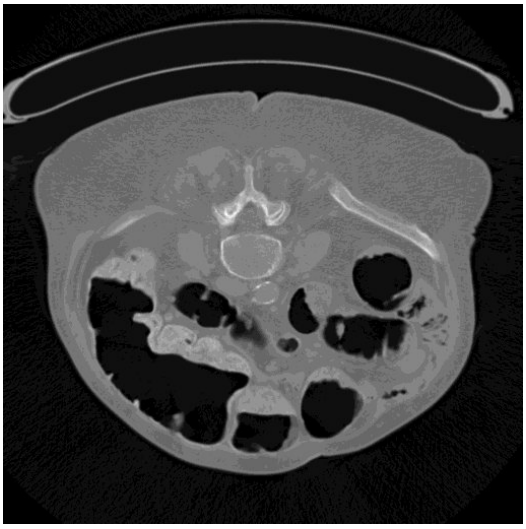
(a) imagen original



(b) imagen sin conocer los 2 últimos bits



(c) imagen sin conocer los 6 últimos bits



(d) imagen sin conocer los 8 últimos bits

Figure 6.3: Ejemplo visual recuperación imágenes.

la información que nos aporta un mensaje utilizamos el concepto de entropía. La entropía indica la cantidad de bits por símbolo necesarios para representar esa información. Este concepto ya lo hemos introducido anteriormente pero ahora vamos a ver como se calcula.

La entropía de una señal X se define como:

$$\mathrm{H}(X) = -\sum_i p(X_i)\ log_2\ p(X_i), \tag{6.7}$$

donde $p(X_i)$ es la probabilidad del símbolo i.

Por ejemplo, supongamos que queremos codificar el mensaje "ABAB", en este caso tenemos un mensaje de 4 caracteres de 2 símbolos distintos la "A" y la "B". La probabilidad de A en el mensaje es $P(\mathrm{A}) = \frac{2}{4} = \frac{1}{2}$ y la de "B" lo mismo $P(\mathrm{B}) = \frac{2}{4} = \frac{1}{2}$. La entropía para este mensaje seria de

$$\mathrm{H}(X) = -(\frac{1}{2}log_2\frac{1}{2} + \frac{1}{2}log_2\frac{1}{2}) = 1 \text{ bits por sample (bps)}, \tag{6.8}$$

por lo que podemos codificar cada símbolo con 1 bit, (1 bps). De modo que la entropía nos indica los bits necesarios para codificar cada símbolo. En este caso, si asignamos un 0 a la "A" y un 1 a la "B" el mensaje codificado seria 0101. De esta forma hemos codificado el mensaje ABAB en 0101, pasando de necesitar 8 * 4 = 4 bytes = 32 bits, a sólo 4 bits.

6.5 Redundancia y compresión

La **redundancia** es el principio básico de la compresión. Para ser capaces de poder codificar de forma que se pueda representar la misma información con menos datos es necesario encontrar la redundancia y explotarla de alguna forma. De forma muy simple podemos entender la redundancia como **porciones del mensaje predictibles a partir de porciones anteriores**.

Supongamos que tenemos la siguiente secuencia de caracteres a codificar:

AAAAAAAAAA BBBBBBBBBB XXXXXXXXXX TTTTTTTTTT

en este caso tenemos un total de 43 caracteres 10 As, 10 Bs, 10 Xs, 10 Ts y 3 espacios en blanco. Cada carácter se almacena en 1 byte de modo que necesitamos 43 bytes, y estamos utilizando un total de 8 bps (bits per sample).

Uno de las primeras técnicas de compresión fué el **Run Length Encoding** (RLE) [5]. El RLE tanto se utiliza para texto como imágenes o vídeos. El RLE consiste, básicamente en substituir secuencias de caracteres por un único carácter seguido del número de repeticiones. Para la secuencia anterior tendríamos:

10A *10B* *10X* *10T*

donde los * se utilizan como elementos separadores de las palabras código. A modo de ejemplo *10A* nos indica que tenemos 10 As consecutivas, seguidas de un espacio en blanco, a continuación 10 Bs consecutivas, seguidas de otro espacio en blanco, y así sucesivamente. En este caso, observamos que con sólo 20 bytes podemos codificar la secuencia original. 20 bytes / 43 muestras = 3,72 bps, mucho menor que 8 bps.

Ahora bien que pasaría si tuviésemos la siguiente secuencia a codificar:

AA AA AA AA AA BB BB BB BB BB XX XX XX XX XX TT TT TT TT TT

esta secuencia esta formada por 69 caracteres y su codificación con RLE no daría la siguiente salida:

2A *2A* *2A* *2A* *2A* *2B* *2B* *2B* *2B* *2B* *2X* *2X* *2X* *2X* *2X* *2T* *2T* *2T* *2T* *2T*

En este caso se puede observar fácilmente que el RLE en lugar de comprimir lo que hace es expandir, es decir necesitamos más bytes para representar la información con la codificación de el RLE que si no la hubiésemos codificado. Después del RLE obtenemos una secuencia que requiere de 99 bytes. 99 bytes / 69 muestras = 11,47 bps que es mayor a 8 bps, de modo que estamos expandiendo!!.

Fácilmente se puede observar que el problema del RLE es cuando encuentra muchas palabras y no ha una secuencia seguida de caracteres. Para evitar estos problemas en 1984 Terry Welch desarrolla el LZW (Lempel-Ziv-Welch) [5] en donde se define un diccionario que busca de secuencias de símbolos (busca patrones), sustituyendo las palabras aprendidas por referencias al diccionario. Consideremos nuevamente la secuencia a codificar:

AA AA AA AA AA BB BB BB BB BB XX XX XX XX XX TT TT TT TT TT

la ejecución del algoritmo de codificación LZW da como salida:

AA *[12][3]*BB *[12][3]*XX *[12][3]*TT *[12][3]*

otra vez, los * se utilizan como elementos separadores de las palabras código. A modo de ejemplo “AA *[12][3]*” nos indica que a partir del espacio en blanco retrocedemos 3 posiciones y ponemos 12 copias de los 3 siguientes caracteres, obteniendo como resultado:

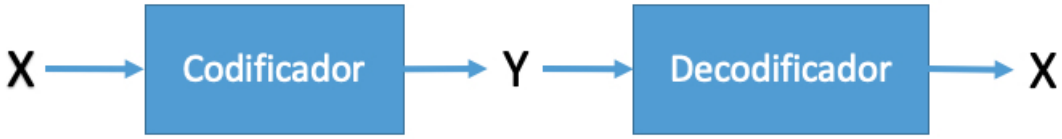


Figure 6.4: Esquema simple codificación y decodificación.

AA AA AA AA AA BB *[12][3]*XX *[12][3]*TT *[12][3]*

trivialmente podemos repetir la operación para BB, XX y TT, obteniendo la secuencia original. en este caso hemos codificado la secuencia original utilizando 44 bytes, por lo que los bps = 44 bytes * 8 bits / 69 muestras = 5,10 bps!

Un aspecto que nos debe quedar muy claro es que estos codificadores **RLE y LZW no introducen ningún tipo de pérdida de información, recuperando los datos originales que entran en el codificador**. De aquí en adelante cuando hablemos de codificadores, haremos referencia a técnicas que permiten recuperar los datos originales. La figura 6.4 nos muestra un esquema simple del proceso de codificar una señal X, obtener unos datos codificados Y y a continuación decodificar los datos Y obteniendo los originales X.

6.5.1 Redundancia en imágenes digitales

Hasta este momento hemos visto como podemos explotar la redundancia aplicando una técnica de codificación aplicado a secuencia de caracteres. Pero como bien sabemos, las imágenes y vídeos digitales se estructuran en muestras/píxeles (ver 6.2 y cada uno de estos muestras son valores numéricos. Ahora, nos debemos preguntar si podemos aplicar antes de la codificación alguna técnica que nos permita reducir la información a codificar? Al tratarse de imágenes, la respuesta es Sí, y vamos a verlo mediante otro sencillo ejemplo.

La idea principal de la compresión de imágenes se basa en que los píxeles vecinos están altamente correlacionados, es decir, que son muy parecidos. Esta correlación se la conoce como correlación espacial. Supongamos que tenemos los siguientes muestras de una imagen digital X:

12, 17, 14, 19, 21, 26, 23, 29, 41, 38, 31, 44, 46, 57, 53, 50, 60, 58, 55, 54, 52, 51, 56, 60

en este caso las técnicas de RLW y LZW no serian efectivas ya que no existen secuencias repetidas, notase que:

- sólo 2 muestras tienen el mismo valor,
- y la entropía de $H(X) = -(\frac{2}{24}log_2\frac{2}{24} + 22(\frac{1}{24}log_2\frac{1}{24})) = 4,51$bps

6.5.2 Predicción

La primera técnica para explotar la correlación espacial que vamos a ver va a ser la predicción. Veamos esta técnica mediante un sencillo, pero ilustrativo, ejemplo. Si calculamos la diferencia de los muestras adyacentes, 17 - 12, 14 - 17, 19 - 14, 21 - 19, etc. Obtendremos la unos nuevos datos X'

12, 5, -3, 5, 2, 5, -3, 6, 12, -3, -7, 13, 2, 11, -4, -3, 10, -2, -3, -1, -2, -1, 5, 4,

Es importante destacar que en este caso lo que se hace es realizar una predicción P y calcular un residuo X'. Siendo X el vector de los datos originales e X_i el valor de X en la posición i del vector, entonces la predicción $P_i = X_{i-1}$. En este caso, estamos "suponiendo" que el siguiente sample será muy parecido al actual de modo que el valor resultante sera próximo al cero y constante, obteniendo una entropía $H(X') < H(X)$. Después de aplicar la predicción podemos observar que:

- existe 1 secuencia de 5 valores iguales (-3, -3, -3, -3, -3,),
- existe 1 secuencia de 4 valores iguales (5, 5, 5, 5),
- existe 4 secuencias de 2 valores iguales (-2, -2), (-1, -1), (2, 2) y (12, 12),
- existe 7 valores únicos (-7, -4, 4, 6, 10, 11, 13),
- de modo que la entropía de $H(X') = -((\frac{5}{24}log_2\frac{5}{24}) + (\frac{4}{24}log_2\frac{4}{24}) +$

  $+ 4(\frac{2}{24}log_2\frac{2}{24}) + 7(\frac{1}{24}log_2\frac{1}{24})) = 3,44$bps

A este proceso se le conoce como decorrelación espacial. En la literatura existen distintas técnicas de decorrelación espacial las cuáles se pueden clasificar como técnicas **predictivas** o basadas en **transformada**. La que acabos de

ver es una técnica basada en predicción, ya que lo que hacemos es predecir el sample siguiente mediante el anterior mediante:

$$X'_i = \begin{cases} X'_0 = X_0 & \text{si } i = 0, \\ X_i - X_{i-1} \text{ si } i > 0, \end{cases} \tag{6.9}$$

De modo que, este es un proceso fácilmente reversible utilizando la siguiente función:

$$X_i = \begin{cases} X_0 = X'_0 & \text{si } i = 0, \\ X'_i + X_{i-1} & \text{si } i > 0, \end{cases} \tag{6.10}$$

Para determinar si un predictor funciona de forma eficiente se puede utilizar la entropía de la señal de resultante de la predicción X' o también se puede utilizar la Sum of Absolute Differences (SAD)

$$\text{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} |X'_{i,j}|, \tag{6.11}$$

como menor sea el valor de SAD nos indicara que el predictor realiza una mejor predicción de la señal original X.

6.5.3 Transformada Wavelet

Por otro lado, y como hemos comentado tenemos los métodos basados en transformada, los cuales utilizan formulación matemática para transformar los datos originales X en X' y a posteriori codificar X'. Las transformadas wavelet se han hecho un hueco muy importante en los sistemas de compresión, debido a su alta capacidad de decorrelación y la descomposición por niveles de resolución. El hecho de obtener distintos niveles de resolución permiten descomprimir estos niveles de forma independiente, aportando un gran versatilidad a la hora de transmitir y/o descomprimir de forma parcial una imagen de grandes dimensiones. La figura 6.5 muestra la aplicación de una descomposición de 3 niveles de una transformada wavelet. La transformada wavelet transforman la señal de manera que la divide en dos partes, una parte contiene los datos a una menor resolución, denotada como L. Mientras que la otra parte contiene los detalles, denotados como H, y son necesarios para recuperar los datos originales.

Una de las transformadas wavelet más simples es la transformada de Haar [6]. Los datos transformados a baja resolución L se obtienen mediante la ecuación 6.12, y los detalles H se obtienen a través de la ecuación 6.13.

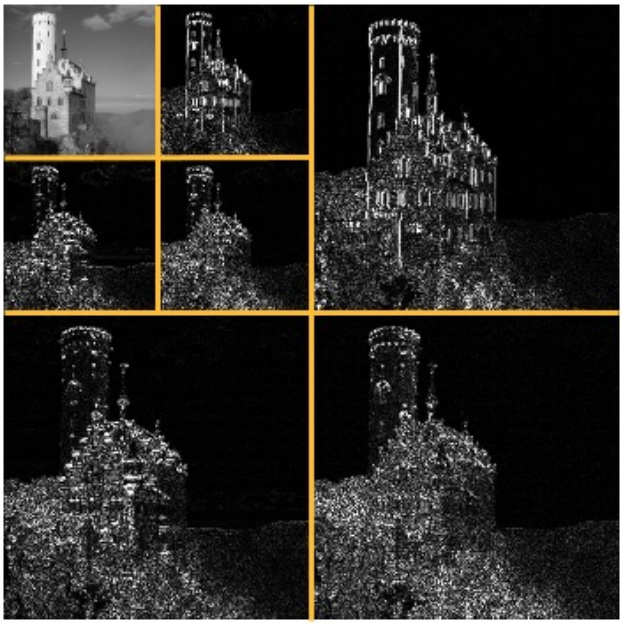


Figure 6.5: Ejemplo de transformada wavelet de 3 niveles de resolución

$$L_{2n}^{j+1} = \frac{L_{2n}^{j} + L_{2n+1}^{j}}{2} \tag{6.12}$$

$$H_{2n}^{j+1} = \frac{L_{2n}^{j} - L_{2n+1}^{j}}{2} \tag{6.13}$$

La Figura 6.6 muestra un ejemplo de aplicación de la transformada Haar en una descomposición de 3 niveles para un vector de datos

$$X = [\,1, 2, 3, 4, 5, 6, 7, 8\,],$$

obtenemos la siguiente señal transformada

$$X' = [\,4.5, -2, -1, -1, -0.5, -0.5, -0.5, -0.5]$$

Vamos a ver un ejemplo pasos que se aplican en cada uno de los niveles:

- Inicialmente tenemos la señal original X, después de aplicar un primer nivel del filtro obtendremos $L1$ y $H1$. Nos encontramos en $j = 0$, para calcular los valores de $j = 1$ y $n = 0$, siendo n la posición del vector X que se va actualizando. Las operaciones para calcular L_0^1, L_1^1, H_0^1, H_1^1 son:

$$L_0^1 = \frac{1+2}{2} = 1.5 \ , L_1^1 = \frac{3+4}{2} = 3.5 \tag{6.14}$$

$$H_0^1 = \frac{1-2}{2} = -0.5 \ , H_1^1 = \frac{3-4}{2} = -0.5 \tag{6.15}$$

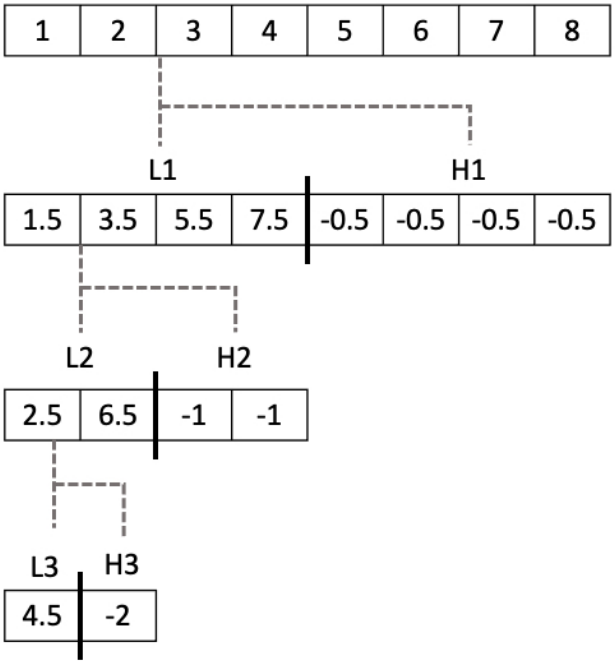


Figure 6.6: Ejemplo de descomposición de transformada Haar en 3 niveles de resolución

- Sobre la señal $L1$ podemos volver a aplicar un nivel de transformada, obteniendo el nivel de resolución 2 y sus respectivos detalles, denotados como $L2$ y $H2$.

Es importante mencionar que la señal que se debe almacenar al final es $L3, H3, H2, H1$ ya que a partir de $L3$ y $H3$ podremos recuperar $L2$, a partir de $L2$ y $H2$ podremos recuperar $L1$ y junto con $H1$ tendremos X. si deseamos recuperar $L2$ tendremos:

$$L_0^2 = \frac{4.5 + -2}{=} 2.5 \ , L_1^2 = \frac{4.5 - -2}{=} 6.5 \tag{6.16}$$

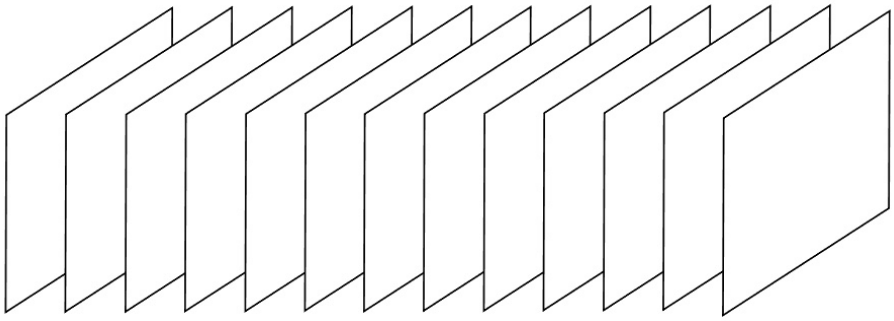


Figure 6.7: Ejemplo de frame rate con fps = 13.

De modo que este es un proceso reversible, el cual nos permite ir recuperando los datos a distintos niveles de resolución. Ahora bien, en este ejemplo hemos aplicado la transformada Haar sobre un vector, pero las imágenes digitales son matrices (vectores de 2 dimensiones). En el caso de imágenes digitales, se aplica primero un nivel de transformada wavelet sobre el eje horizontal y a continuación sobre el eje vertical. Si se desean más niveles se vuelve a realizar la operación sobre la señal L. La figura 6.6 muestra un ejemplo de aplicación de la transformada Haar en una imagen, en la cual se ha realizado una descomposición de 3 niveles. En el mundo de la compresión se han introducido distintas transformadas wavelets, han que tener presente que algunas introducen pérdida y otras NO, permitiendo recuperar los datos originales.

Las dos técnicas vistas para explotar la redundancia espacial se aplican para la codificación de imágenes digitales. Cabe destacar que si deseamos comprimir un vídeo existe también mucha redundancia entre los frames consecutivos. Un frame, es una imagen digital en un vídeo, el número de imágenes que visualizamos por segundo viene determinado por el "frame rate". Lo más habitual es tener un frame rate por segundo fps = 24, de modo que cada segundo tenemos 24 imágenes digitales, esta densidad de frames por segundo hace que frames contiguos sean muy parecidos, a menos que haya movimientos bruscos (como en los deportes). La Figura 6.7 muestra un ejemplo esquemático de 13 frames capturados en un segundo, produciendo una secuencia de vídeo con un fps = 13, donde cada rectángulo representa un frame.

La reducción de la correlación entre los distintos frames las podemos hacer mediante 4 técnicas distintas:

6.5.4 Subsampling

Durante la adquisición de imágenes de un vídeo, dependiendo del movimiento efectuado por los objetos que aparecen en el vídeo, se puede dar el caso que la diferencia en frames consecutivos sea muy reducida. Subsampling se basa en considerar que si hay muy poco movimiento los frames consecutivos serán altamente parecidos, de modo que se pueden descartar algunos frames. En la Figura 6.8 se muestra un ejemplo de un vídeo con un subsampling de 2, es decir cada 2 frames uno es descartado. Esto tiene lugar durante el proceso de compresión, evidentemente el descodificador deberá volver a generar los frames descartados –los azules– utilizando los frames adyacentes –los blancos–.

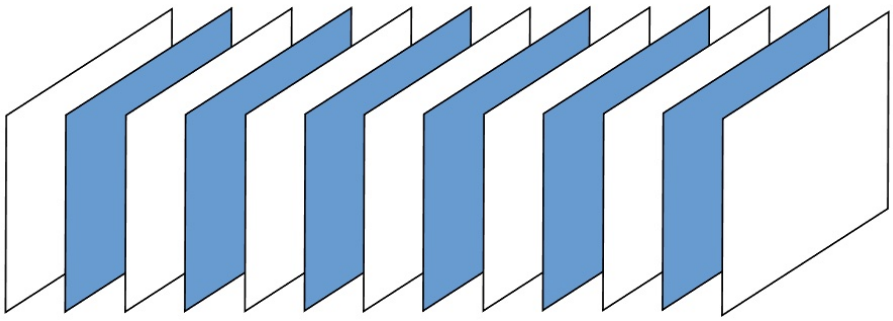


Figure 6.8: Ejemplo de subsampling

6.5.5 Diferencia de bloque

Mientras que la técnica del subsampling se basa en considerar que frames consecutivos son prácticamente iguales, la diferencia de bloque considera que frames consecutivos son mayoritariamente muy parecidos. Por eso motivo la diferencia de bloque codifica (opcionalmente) diferencias de una misma zona espacial. Primeramente se dividen los frames en bloques (B) –esta estructura puede ser rectangular–. Cada bloque B del frame f_i se compara con el f_{i+1}. Si la diferencia, utilizando la métrica de SAD es pequeña, el bloque se codifica indicando sus coordenadas y la sus diferencias sample a sample con su bloque correspondiente, obteniendo R_B, caso contrario se codifica f_{i+1}. Matemáticamente se describe de la siguiente forma:

$$R_B = \begin{cases} f_{i+1} - f_i & \text{si } SAD < T = 0, \\ f_{i+1} & \text{si } SAD \geq T, \end{cases} \tag{6.17}$$

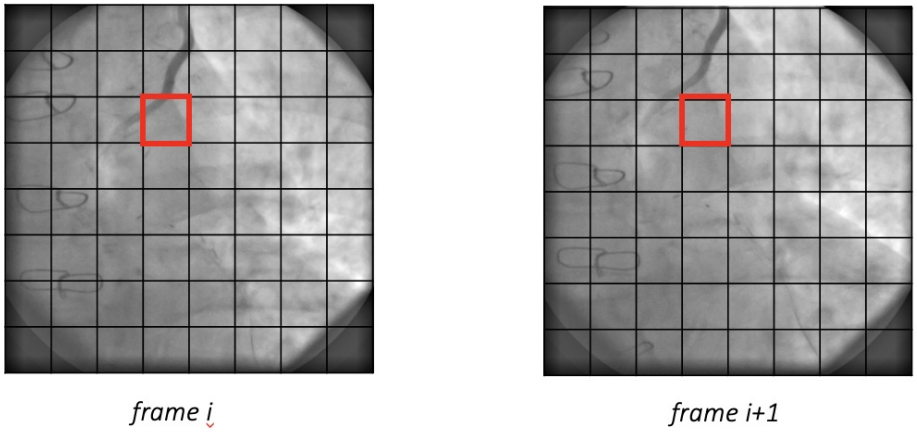


Figure 6.9: Ejemplo de diferencia de bloque

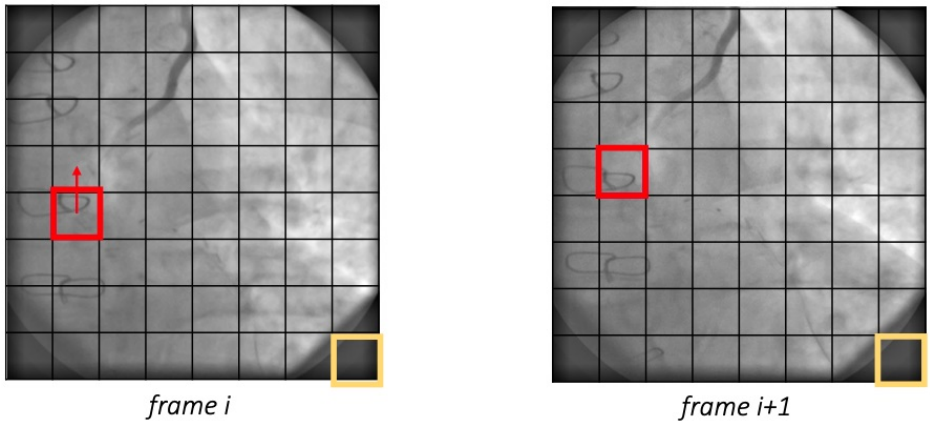


Figure 6.10: Ejemplo de compensación de movimiento

donde T es un threshold determinado por el usuario especialista en compresión de datos. La figura 6.9 muestra dos frames con una subdivisión por bloques lógica, donde se resalta (en rojo) los bloques de una misma región espacial que deben analizarse para decidir si se codifican las diferencias o los muestras de f_{i+1}

6.5.6 Compensación de movimiento

En una película, puede haber una translación de todo el frame en horizontal, vertical o diagonal. La compensación de movimiento pretende utilizar esta

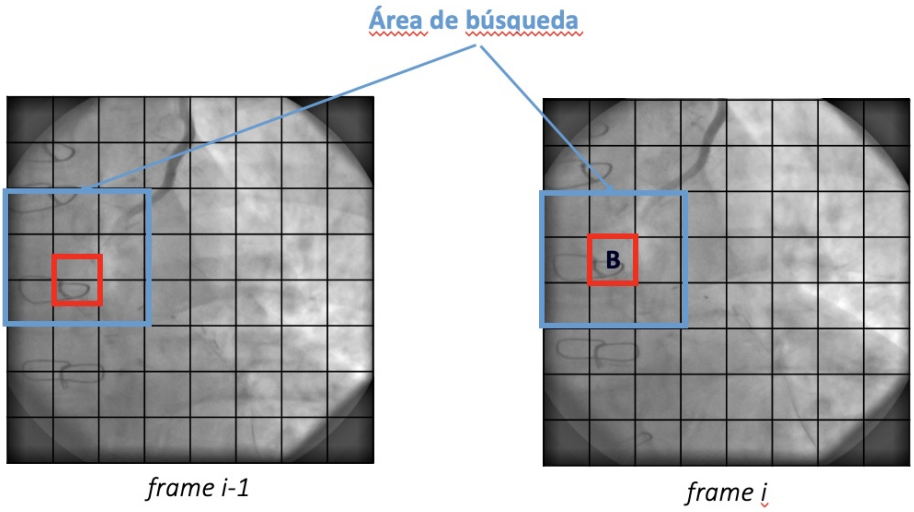


Figure 6.11: Ejemplo de búsqueda de bloque

translación en todo el frame codificando las diferencias entre los bloques, siempre que estos sean parecidos. En el momento de codificar es necesario almacenar también el vector que nos indica el desplazamiento que ha hecho el bloque, de este modo podremos recuperar la información correctamente. En la compensación de movimiento, sólo almacenamos un vector por frame, considerando que todos los bloques se les aplica el mismo desplazamiento.

6.5.7 Búsqueda de bloque

Esta técnica sume un mayor movimiento que el considerado por al diferencia de bloque. Supongamos una película, en esta se puede observar que la diferencia entre fotogramas consecutivos la diferencia es mínima, debido a que sólo se mueven algunos objetos de la escena, la cámara o ambos a la vez. El objetivo de esta técnica es buscar donde se han movido los objetos entre fotogramas consecutivos. Para ello, primeramente necesitamos realizar una subdivisión lógica de los frames en bloques B_j. Para cada bloque B_j del frame f_{i+1}, debemos buscar el bloque más parecido a el minimizando la función

$$MIN(SAD(B_j^{f_{i+1}}, B_n^{f_i})\forall j, \ , \tag{6.18}$$

donde $B_n^{f_i}$ es el bloque n del frame f_i a codificar y $B_j^{f_{i-1}}$ son todos los posibles bloques a comparar del frame anterior. La figura 6.10 muestra un

ejemplo gráfico de la técnica de compensación de movimiento, donde el bloque B resaltado en rojo del frame f_i es el bloque a codificar. Es fácilmente observable que realizar dicha búsqueda puede llevar mucho tiempo, en función del tamaño del frame y los bloques. Considerando que entre frames consecutivos los objetos no se habrán movido grandes distancias, esto permite reducir la era de búsqueda y así reducir el tiempo de compresión. Para ello se define el "Área de busqueda" como una región adyacente al bloque a codificar donde realizaremos la búsqueda. En la figura 6.11 se muestran los dos frames el bloque a codificar y la área de búsqueda.

Bibliography

[1] https://en.wikipedia.org/wiki/Pigeonhole_principle. Accessed: 2021-09-01.

[2] https://es.wikipedia.org/wiki/Byte. Accessed: 2021-09-01.

[3] https://en.wikipedia.org/wiki/ASCII. Accessed: 2021-09-01.

[4] https://en.wikipedia.org/wiki/Entropy_(information_theory). Accessed: 2021-09-01.

[5] https://es.wikipedia.org/wiki/LZW. Accessed: 2021-09-01.

[6] https://en.wikipedia.org/wiki/Haar_wavelet. Accessed: 2021-09-01.

Chapter 7

Compresión de Datos II

Joan Bartrina

7.1 Métricas de distorsión

La calidad de imagen puede degradarse debido a distorsiones introducidas durante el proceso de compresión. La comunidad científica ha definido distintas medidas analíticas con el fin de medir la calidad las imágenes.

En nuestro escenario disponemos de los datos originales, por ejemplo la imagen sin distorsión, hecho útil ya que permite utilizarla como referencia para medir la calidad de la imagen comprimida y posteriormente descomprimida. De modo que podremos evaluar la calidad de las imágenes comprimidas, una versión sin comprimir de la imagen proporciona una referencia útil. En estos casos, puede utilizar métricas de calidad de referencia completa para comparar directamente la imagen de destino y la imagen de referencia.

Los algoritmos de referencia completa comparan la imagen de entrada con una imagen de referencia sin distorsión. Las medidas de distorsión más comunes son Mean Square Error (MSE), el Peak Signal Noise Ratio (PSNR) y el Peak Absolute Error (PAE). A continuación vamos a ver como se calculan cada una de estas dos medidas de distorsión y calcularemos un ejemplo.

7.1.1 Mean Squared Error

En procesamiento de imágenes, el Mean Square Error (MSE) es una métrica que mide la fidelidad entre los datos originales X y los comprimidos y posteriormente descomprimidos, es decir, los datos recuperados X'. El MSE mide el promedio de los errores al cuadrado, es decir, calcula la diferencia entre el valor original X y el recuperado X', y esa diferencia es elevada al cuadrado.

Supongamos que $X = \{X_{i,j} | i = 1, 2, ..., M \wedge j = 1, 2, ..., N\}$ y $X' = \{X'_{i,j} | i = 1, 2, ..., M \wedge j = 1, 2, ..., N\}$ son dos señales discretas finitas, en nuestros casos imágenes visuales, donde M es el número de muestras de la señal (píxeles, cuando las señales son imágenes), mientras que $X_{i,j}$ y $X'_{i,j}$ son los valores de X y X', respectivamente. El MSE entre dos señales se calcula mediante la siguiente expresión

$$\text{MSE}(X, X') = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (X_{i,j} - X'_{i,j})^2. \tag{7.1}$$

De esta expresión se puede observar fácilmente que si las dos señales X y X' (imágenes) son iguales el valor de MSE es 0. Por el otro lado, como mayor sea al valor del MSE más distorsión entre X y X' habrá.

7.1.2 Peak Signal Noise Ratio

El Peak Signal Noise Ratio (PSNR) es una métrica derivada del MSE. En este caso relaciona la máxima energía posible de la señal y el ruido –el MSE– que afecta a su representación. Debido a que muchas señales tienen un gran rango dinámico, el PSNR se expresa generalmente en escala logarítmica. Como unidades se utiliza el decibelio (dB). El uso más habitual del PSNR es como medida cuantitativa de la calidad de la reconstrucción de imágenes comprimidas. Para su calculo, es necesario calcular previamente el MSE entre dos señales X y X'. El PSNR se define como:

$$\text{PSNR}(X, X') = 10 Log_{10} \frac{L^2}{\text{MSE}}, \tag{7.2}$$

donde L es el rango dinámico. Por ejemplo, para imágenes donde cada muestra se presenta con 8 bits por muestra, $L = 2^8 - 1 = 255$. Para imágenes de 12 bits por muestra, aunque se almacenen en 2 bytes (16 bits), su rango dinámico es $L = 2^{12} - 1 = 4095$.

En este caso, y a diferencia del MSE, si el PSNR es ∞ indica que las señales X y X' son iguales, mientras que un valor más próximo al 0 indica que las imágenes son más distintas.

7.1.3 Peak Absolute Error

La última métrica interesante en los procesos de compresión es el Peak Absolute Error (PAE) que no indica cual es el error máximo en valor absoluto que se esta cometiendo entre la señal original X y la recuperada X' y se expresa de la siguiente forma

23	25	1790	123	125
230	125	789	124	156
231	235	1432	173	185
233	225	1290	134	124
233	215	1490	236	122

Tabla 7.1: Señal X

22	24	1789	122	124
228	124	788	122	154
230	234	1430	172	184
232	224	1288	132	122
232	214	1488	234	120

Tabla 7.2: Señal X'

$$\text{PAE}(X, X') = \text{MAX}(|X - X'|) \tag{7.3}$$

7.1.4 Ejemplos

En esta subsección vamos a calcular las métricas descritas anteriormente, el MSE, PSNR y PAE entre dos señales X y X'. La tablas 7.1 y 7.2 contienen los datos utilizados como señales X y X', respectivamente, y que serán utilizadas para el calculo de las métricas descritas anteriormente en modo de ejemplo.

1. **Calculo MSE**:

   $\text{MSE}(X, X') = \frac{1}{5*5}((23-22)^2 + (25-24)^2 + (1790-1789)^2 + (123-122)^2 + (125-124)^2 ... (236-234)^2 + (122-120)^2) = \frac{55}{25} = 2,2.$

2. **Calculo PSNR**:

   Para el calculo del PSNR debemos primero averiguar el rango dinámico de los datos. Para ello buscamos el valor máximo en X, MAXIMO(X) = 1790. Calculamos los bits necesarios para representar el 1790 en binario mediante $\lceil Log_2(1790) \rceil = 11$ bits.

$\text{PSNR}(X, X') = 10Log_{10}\frac{2^{11}-1}{55} = 10Log_{10}\frac{2047}{55} = 48,81\text{dB}.$

3. **Calculo PAE**:

   $\text{PAE}(X, X') = \text{MAX}(|23-22|, |25-24|, |1790-1789|, |123-122|, |125-124|, ..., |236-234|, |122-120|) = 2$, indicando que el error máximo que se esta cometiendo entre los datos originales y los recuperados es de 2 unidades.

7.2 Compresión lossless y lossy: el pipeline

Como hemos visto anteriormente los métodos de compresión se pueden clasificar en dos categorías principales, métodos de compresión sin pérdida (**lossless**) y con pérdida (**lossy**). Además cada uno de estos compresores pueden utilizar técnicas de transformada o de predicción para explotar la redundancia de la imagen y así reducir su entropía.

Las **técnicas lossy** son aquellas que no recuperan los datos originales, es decir, introducen algún tipo de pérdida durante el proceso de compresión que no se puede recuperar. El hecho de introducir esta pérdida nos permite obtener unos factores de compresión mucho más elevados, eso sí, penalizando en como se recuperan los datos, tal y como hemos visto en la figura 6.3. La pérdida durante el proceso de compresión se introduce o bien utilizando utilizando técnicas de **cuantización** o bien utilizando técnicas de **rate control**.

7.2.1 Cuantización

Al introducir pérdida las técnicas de compresión se aprovechan de que el ojo humano es bastante bueno percibiendo las pequeñas diferencias en el brillo sobre un área relativamente extensa, pero no es tan bueno distinguiendo la misma variación de intensidad entre muestras cercanas. Este echo nos permite poder eliminar estas pequeñas diferencias entre valores. Para ello utilizaremos un cuantizador.

$$\hat{X} = sign(X)\left\lfloor \frac{|X|}{\Delta} \right\rfloor, \tag{7.4}$$

donde X son los valores originales, $\lfloor\ \rfloor$ permite obtener sólo la parte entera (descartando los decimales), $|X|$ devuelve el valor absoluto de X, $sign(X)$ nos indica el signo del valor X, y Δ nos indica el paso de cuantización. Si $\Delta = 1$, entonces $X = \hat{X}$, de modo que no se está introduciendo pérdida durante el proceso. Mientras que, lado como mayor sea el valor de Δ los

valores de $\hat{X}$ serán más cercanos a cero y más homogéneos. Al ser más homogéneos la entropía es menor y en consecuencia los bits necesarios son menores.
Vamos a ver un breve ejemplo utilizando datos previamente utilizados. Consideramos de nuevo los datos originales X

12 17 14 19 21 26 23 29 41 38 31 44 46 57 53 50 60 58 55 54 52 51 56 60

con una entropía de $H(X) = 4,51$. Si cuantizamos X' utilizando 7.5 con $\Delta = 5$ obtendremos $\hat{X}'$

2 3 2 3 4 5 4 5 8 7 6 8 9 11 10 10 12 11 11 10 10 10 11 12

donde la de entropía $H(\hat{X}') = -(4 \cdot \frac{1}{24} log_2 \frac{1}{24} + 4 \cdot \frac{2}{24} log_2 \frac{2}{24} + \frac{3}{24} log_2 \frac{3}{24} +$

$+ \frac{4}{24} log_2 \frac{4}{24} + \frac{5}{24} log_2 \frac{5}{24}) = 3,22$bps.
Para recuperar los datos debemos aplicar la función inversa del proceso de cuantización, que en este caso se define como:

$$X'' = \Delta \cdot \hat{X}', \tag{7.5}$$

donde obtendremos X'' como

10 15 10 15 20 25 20 25 80 35 30 40 45 55 50 50 60 55 55 50 50 50 55 60

si calculamos donde el PAE$(X, X') = 4$. En este punto es importante comentar que siempre se cumplirá que

$$PAE(X, X') \leq \Delta - 1, \tag{7.6}$$

por lo que si $\Delta = 1$ el PAE$(X, X') = 0$ indicando que los datos originales y recuperados, X y X' son iguales. Si $\Delta > 1$ los datos originales y recuperados serán distintos, obteniendo siempre un PAE$(X, X') = \Delta - 1$.

El uso del cuantizador, seguido de un codificador nos permite controlar el error máximo cometido en X y X', pero no podemos controlar el tamaño del archivo final.

7.2.2 Rate control

Las técnicas de rate control, nos permiten tener un control sobre el tamaño del archivo final, de modo que podemos ajustar el tamaño del archivo al ancho de banda del canal por donde se va a transmitir el archivo. Ahora bien, estas técnicas no permiten controlar el error. De forma general las técnicas de

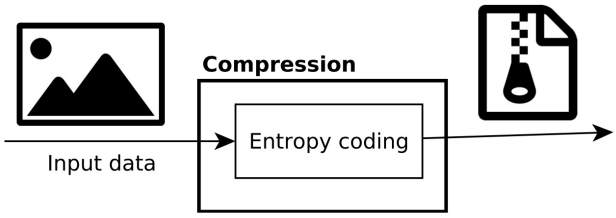


Figure 7.1: Pipeline simple con entropía codificador

rate control, seleccionan cuales son los conjuntos de datos codificados que permiten recuperar las imágenes a una mejor calidad con un determinado rate. El rate, se le conoce como el valor de bps que permite definir el tamaño del archivo comprimido.

El pipeline de un sistema de compresión se le conoce como el conjunto de técnicas que se aplican de forma secuencial que permite comprimir las imágenes. La combinación de estas técnicas permite definir distintos pipelines de compresión con determinadas características. A continuación vamos distintos pipelines en función de las técnicas incluidas en ellos.

Pipeline basado en entropía codificador

Permite reducir los datos representando exactamente la misma información. En la literatura podemos encontrar distintos codificadores por entropía como LZ77, LZ78, MQ, entre otros. La figura 7.1 muestra un un método de pipeline simple formado únicamente por un entropía codificador, por lo que si se aplica de forma inversa permite recuperar los datos originales sin pérdida de información.

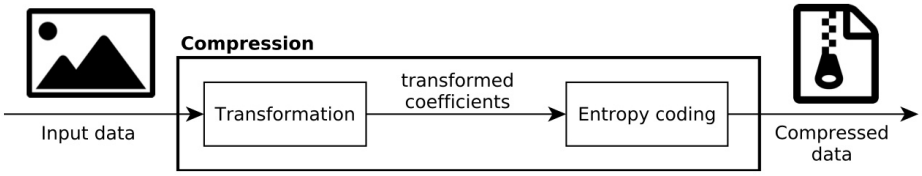


Figure 7.2: Pipeline formado por una transformada y un entropía codificador

Pipeline basado en transformada

Como vimos en la unidad anterior las transformadas permiten reducir la entropía de la imagen a codificar. En nuestro caso vimos la transformada wavelet HAAR. Las transformadas wavelet se aplican sobre la imagen, y a continuación los datos transformados suelen ser codificados con mediante un entropía codificador. La figura 7.2 muestra un esquema con estas características. Para recuperar la imagen original debemos deshacer el proceso de forma inversa, de modo que primero utilizaremos un entropía descodificador y a continuación aplicaremos la transformación inversa para se recuperar los datos originales.

Pipeline: Predicción

En un pipeline basado en la predicción obtenemos unos valores residuales X', como vimos en la unidad 6, estos valores a continuación se deben codificar con un codificador por entropía. La figura 7.3 muestra un pipeline basado en la predicción, es importante destacar que en el proceso de compresión el predictor obtiene los residuales X' utilizando el operador resta, mientras que en el descodificador para recuperar la señal original X se utiliza el operador suma.

Pipeline basado en cuantización y entropía codificador

El pipeline de la figura 7.4 esta formado por un cuantizador y un codificador por entropía. En este caso los datos originales X son cuantizados obteniendo X', posteriormente los datos X' son enviados al codificador por entropía. Esto implica que los datos recuperados al aplicar el descodificador por entropía y el descuantizador no los vamos a recuperar los datos originales, a menos que el paso de cuantización sea de $\Delta = 1$ (fijaos en la fórmula 7.5). De modo que el simple hecho de introducir un cuantizador no implica que no se pueda recuperar la original.

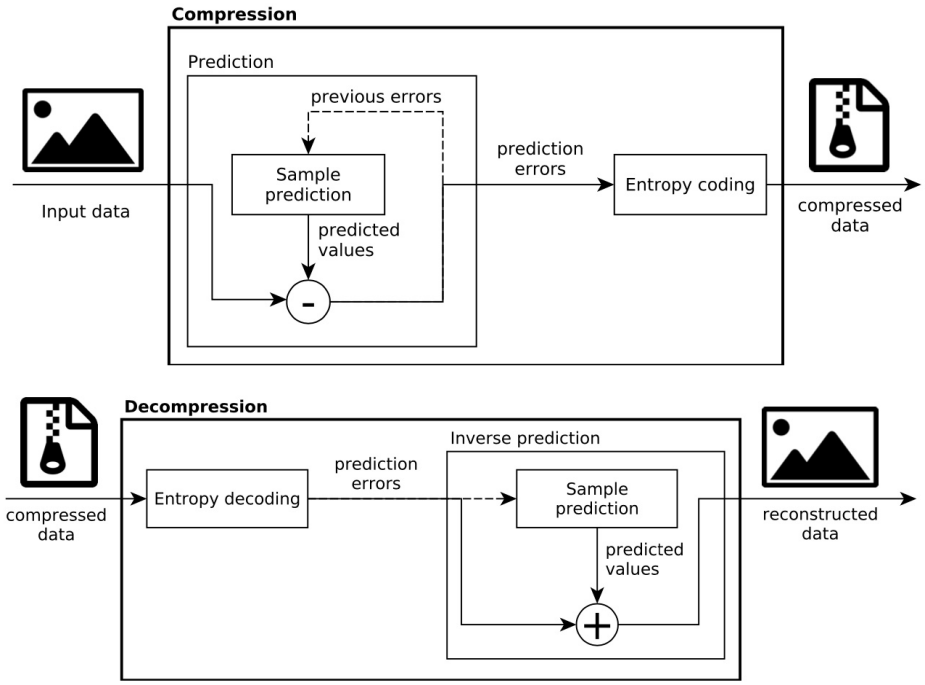


Figure 7.3: Pipeline basado en predicción y entropía codificador

Pipeline basado en transformación, cuantización y entropía codificador

La figura 7.5 muestra un esquema de compresión basado en transformada wavelet seguido de un cuantizador y un entropía codificador define un esquema de sistema de compresión muy utilizado en la compresión de datos. Este pipeline es utilizado como base en distintos métodos de compresión como JPEG y JPEG2000.

Pipeline basado predicción, cuantización y entropía codificador

La figura 7.6 muestra el esquema del compresor de un pipeline compuesto por un predictor, un cuantizador y un codificador por entropía. De esta figura, cabe destacar la flecha que va de la caja del cuantizador a la predicción, la cual es fundamental para recuperar la señal de forma adecuada. Notase, que el predictor utiliza unos datos de entrada para estimar una predicción, esta función de predicción debe ser igual en el compresor que en el descompresor. Ahora bien, si se aplica una cuantización con $\Delta > 1$, entonces los datos recuperados X'' serán distintos a los originales X, de modo que el predictor del

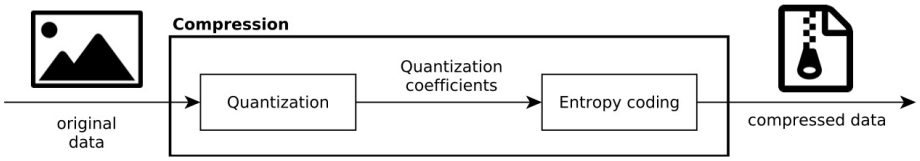


Figure 7.4: Pipeline basado en cuantizador y entropía codificador

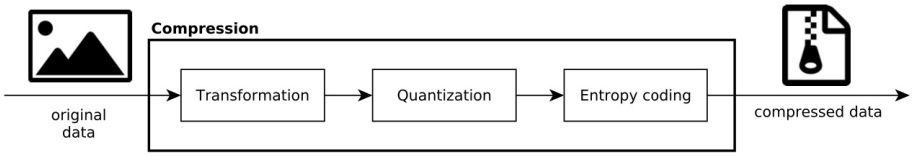


Figure 7.5: Pipeline formado por transformada, cuantizador y entropía codificador

compresor utilizaría X, mientras que el descompresor utilizaría X''. Debido a que la función de predicción es la misma, si los datos son distintos X y X'' el resultado de la predicción también lo será, resultando en un reconstrucción totalmente errónea. Para solucionar este problema, en el proceso de **compresión** se realiza la predicción utilizando los datos cuantizados y descuantizados (simulando los datos que dispondrá el predictor del compresor), de este modo ambos utilizaran X'' para la predicción, lo que permitirá recuperar los datos correctamente.

Los pipelines basado en predicción son muy frecuentes en sistemas de compresión como JPEG-LS, MPEG4/H.264 y HEVC/H.265. Para MPEG4/H.264 y HEVC/H.265 además se incluyen técnicas de subsampling, diferencia de bloque, compensación de movimiento y búsqueda de bloque (vistas en la unidad anterior). Por este motivo JPEG-LS es mucho más rápida que MPEG4/H.264 o HEVC/H.265 durante el proceso de compresión, para la descompresión los tres métodos son muy rápidos, caso contrario no podríamos reproducir el vídeo en tiempo real para MPEG4/H.264 y HEVC/H.265.

7.3 Sistemas de compresión DICOM

En esta sección vamos a ver los sistemas de compresión incluidos en DICOM. No los vamos a ver des del punto de vista técnico, es decir, que técnicas se utilizan para comprimir los datos, sino cuales son sus características para

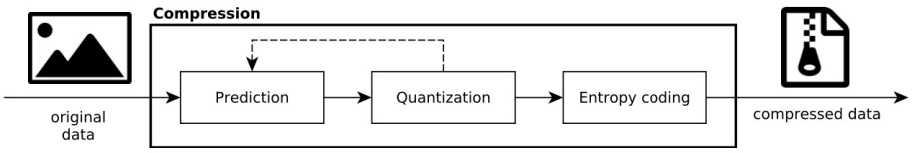


Figure 7.6: Pipeline formado por predicitor, cuantizador y entropía codificador

determinar cuales debemos utilizar en función de los datos a comprimir y su aplicación.

7.3.1 Run Length Encoding

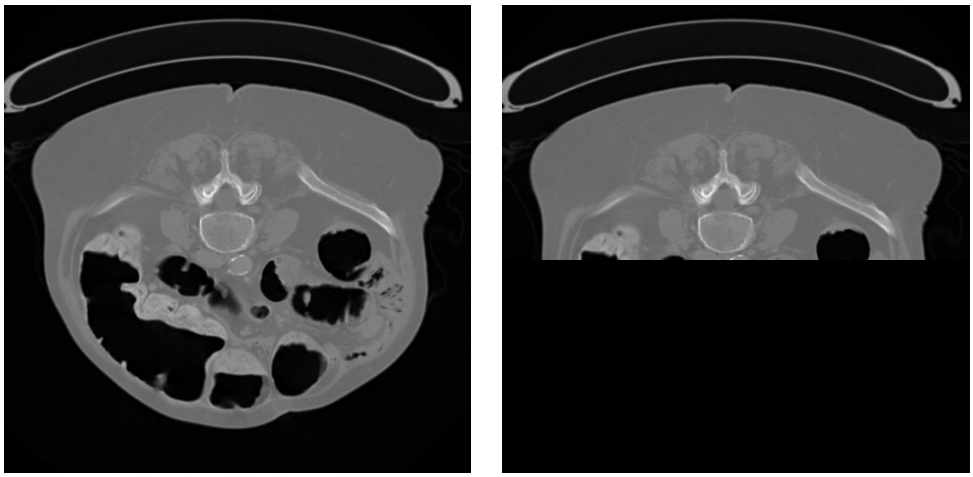
El método de Run Length Encoding (RLE) es un método de codificación sin pérdida descrito en la sección 6.5. Su aplicación es la de comprimir datos sin pérdida de información. En el caso de recuperación parcial del archivo comprimido los datos que se puedan recuperar van a ser sin pérdida de información. Los datos que no se puedan recuperar los vamos a descodificar a un valor fijo, esto dará como resultado un imágen rdonde se visualizan correctamente las N primeras filas y el resto de filas no se van a recuperar. A este efecto se le denomina recuperación *Baseline*. La Figura 7.7 muestra un ejemplo de progresión baseline.

7.3.2 JPEG

Definido por el Joint Photografic Experts Group, el cual le dio nombre al sistema de compresión JPEG. JPEG es sistema de compresión con pérdida. Su uso se encuentra muy extendido en los sistemas fotográficos de uso personal. Para uso médico se tiene que tener en consideración la tasa de compresión. Recordad que altas tasas de compresión no aportaran un impacto sobre la cualidad de la imagen recuperada, de modo que si se realiza un diagnóstico médico este puede verse alterado. Nos permite obtener tasas de compresión de 50 a 1 (la imágen comprimida ocupa 50 veces menos que la original), sin tener mucha pérdida de cualidad visual.

7.3.3 JPEG-LS

También definido por el Joint Photografic Experts Group, este nos permite recuperar los datos sin pérdida de información. Su uso ésde gran interés si



Imágen original Imágen parcialmente recuperada

Figure 7.7: Ejemplo de recuperación parcial de con un sistema de progresión baseline

debemos utilizar las imágenes para un diagnostico médico preciso o utilizando algoritmos de Computed Aided Diagnosis (CAD).

7.3.4 JPEG2000

Un de los recientes estándares presentado por Joint Photografic Experts Group. Permite la compresión con y sin pérdida. Su uso se encuentra extendido a nivel profesional, ya que tiene ciertas características que lo hacen muy interesante para realizar compresiones interactivas. Esto quiere decir que podemos decidir que cantidad de información queremos transmitir/descodificar, a mayor información transmitida/descodificada mayor sera la calidad de la imagen recuperada.

7.3.5 MPEG2, MPEG4/H.264 y HEVC/H.265

Estos tres sistemas de codificación se usa para vídeo. El más común actualmente es el H.264 y H.265. Ambos permiten la codificación sin pérdida de vídeos que capturados con una profundidad de hasta 12 bits por muestra. El problema de estos sistemas es que el tiempo de compresión es muy elevado, pero el de descodificación es rápido.

Bibliography

[1] https://en.wikipedia.org/wiki/Pigeonhole_principle. Accessed: 2021-09-01.

[2] https://es.wikipedia.org/wiki/Byte. Accessed: 2021-09-01.

[3] https://en.wikipedia.org/wiki/ASCII. Accessed: 2021-09-01.

[4] https://en.wikipedia.org/wiki/Entropy_(information_theory). Accessed: 2021-09-01.

[5] https://es.wikipedia.org/wiki/LZW. Accessed: 2021-09-01.

[6] https://en.wikipedia.org/wiki/Haar_wavelet. Accessed: 2021-09-01.