UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

El estándar de compresión y codificación de imágenes del JPEG

Autor/es

ÁLVARO GÓMEZ TROITIÑO

Director/es

ALBERTO ARENAS GÓMEZ y EDGAR LABARGA VARONA

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Matemáticas

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2022-23

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Matemáticas

El estándar de compresión y codificación de imágenes del *JPEG*

Realizado por:

Álvaro Gómez Troitiño

Tutelado por:

Alberto Arenas Gómez

Edgar Labarga Varona

Logroño, julio de 2023

Resumen

Con la llegada de la Revolución digital en la segunda mitad del siglo XX, la tecnología digital ha tomado un papel importante en nuestras vidas cotidianas. En este contexto, la trasmisión de información, particularmente de imágenes, tiene gran relevancia en multitud de tareas que se realizan día a día. Por este motivo, desde hace muchos años se ha tratado de fomentar el avance en técnicas que faciliten la ejecución de estos procesos, tales como la compresión y codificación de imágenes.

Desde sus inicios en 1986 hasta la actualidad, el estándar de compresión y codificación *JPEG* es uno de los métodos más utilizados en imágenes digitales. A lo largo de este trabajo, se tratará de explicar, en la medida de lo posible, todas las matemáticas que rodean a este algoritmo. Para ello, se desarrollarán las diferentes materias sobre las que se fundamenta este procedimiento, entre las que se encuentran el procesamiento de señales, el análisis de Fourier, el tratamiento de color, o los procesos de cuantización y codificación.

Abstract

With the arrival of the Digital Revolution in the second half of the 20th century, digital technology has taken on an important role in our daily lives. In this context, the transmission of information, particularly images, is highly relevant in many tasks that are carried out day by day. For this reason, for many years, efforts have been made to promote advancements in techniques that facilitate the execution of these processes, such as image compression and encoding.

Since its inception in 1986 until today, the *JPEG* compression and encoding standard is one of the most widely used methods in digital images. Throughout this work, an attempt will be made to explain, as far as possible, all the mathematics surrounding this algorithm. To do this, the different subjects on which this procedure is based will be developed, including signal processing, Fourier analysis, color processing, and quantization and encoding processes.

Índice general

1. Introducción **1**
1.1. Procesamiento de señales 4
1.2. Técnicas de compresión de imágenes 5
1.2.1. Tipos de compresión 5
1.2.2. Elementos de compresión 5
1.3. Fundamentos del algoritmo del *JPEG* 7

2. Conceptos elementales del análisis de Fourier **9**
2.1. Series de Fourier 10
2.2. Transformada discreta de Fourier 12
2.3. Algoritmo de la transformada rápida de Fourier 13
2.4. Transformada de Fourier de coseno 22
2.5. Transformada discreta de coseno 23
2.6. Algoritmo de la transformada rápida de coseno 27

3. Compresión del *JPEG* **31**
3.1. Breve historia del *JPEG* 31
3.2. Descomposición en bloques y tratamiento del color 33
3.3. Aplicación de la transformada discreta de coseno 38
3.4. Proceso de cuantización 40
3.5. Proceso de codificación: el código Huffman 42
3.6. Procesos inversos 47

4. Conclusiones **49**

Capítulo 1

Introducción

La compresión de fotografías está muy presente en nuestro día a día. Por ejemplo, un navegador suele cargar numerosas imágenes cuando buscamos una noticia en Internet. También nos encontramos con multitud de ellas cuando entramos en una red social. Sin embargo, los archivos de imágenes normalmente tienen un tamaño digital considerable, lo que provoca que almacenarlos o transferirlos sea una tarea demasiado complicada y, en muchas ocasiones, inviable. En la segunda mitad del siglo XX, se hizo cada vez más evidente la necesidad de disponer de un método de compresión de imágenes común, que facilitara el desarrollo tecnológico. Una solución satisfactoria, y que a la postre sirvió como referencia, la dio el grupo *JPEG* con un estándar de compresión y codificación de imágenes fijas (del inglés *still images*) [17].

Figura 1.1: Imagen de la Catedral de Burgos.

Aunque han aparecido nuevos métodos de compresión en los últimos años, *JPEG* sigue teniendo un papel muy extendido en este ámbito. Ilustremos todo esto muy brevemente con un ejemplo real. Consideremos la fotografía de la Catedral de Burgos de la Figura 1.1. Esta imagen, almacenada en el sistema de ficheros de un ordenador, está compuesta por 1 200 píxeles de ancho por 630 píxeles de alto, que hacen un total de 756 000 píxeles (Figura 1.2).

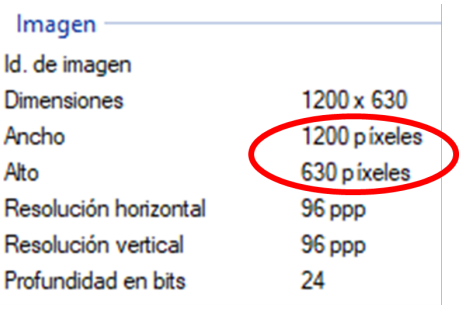


Figura 1.2: Propiedades de la imagen de la Catedral de Burgos.

En principio, cada píxel individual supone la cantidad de tres bytes de memoria: en efecto, la equivalencia 1 píxel–1 byte se tiene si la imagen es en blanco y negro; para imágenes en color son necesarios tres bytes en el modelo *RGB*, uno para el color rojo (*Red*), otro para el verde (*Green*) y un tercero para el azul (*Blue*). De este modo, es de esperar que para guardar la fotografía de la Catedral de Burgos se necesiten 2 268 000 bytes (756 000 × 3 bytes/píxel), o lo que es lo mismo, unos 2 215 KB. Sin embargo, tal y como muestra en la Figura 1.3, si analizamos las propiedades de su archivo gráfico, podemos observar que solamente se requieren algo menos de 110 KB para almacenarla. Esta diferencia notable entre el espacio esperado y el que en realidad ocupa la imagen se debe sin duda a que el archivo gráfico ha sido sometido a un proceso de compresión.

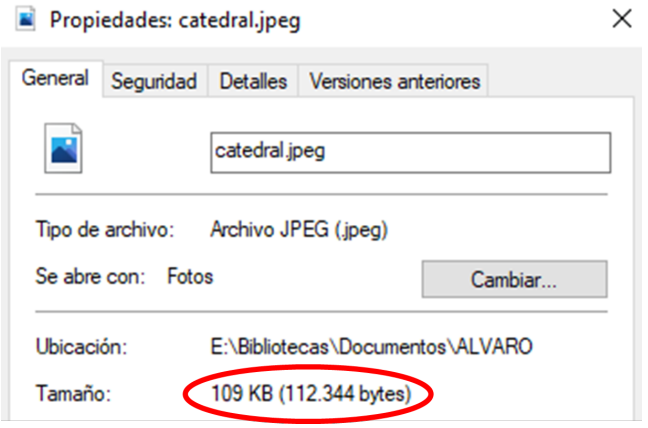


Figura 1.3: Tamaño de almacenamiento de la imagen de la Catedral de Burgos.

En realidad, una imagen en blanco y negro se puede representar en un ordenador por una matriz de números entre 0 y 255, o por tres matrices en el caso de imágenes en color. De esta manera, tal y como vemos en la Figura 1.4, un píxel del cielo de Burgos en la fotografía viene representado por los valores (1, 143, 255) en el espacio *RGB*. Distinguimos en este punto un color azulado porque es precisamente el color azul (255) el que predomina en la mezcla *RGB*.

Esto se puede comprobar si miramos el código de la imagen con un editor de texto. Como se puede apreciar en la Figura 1.5, la imagen de la catedral con el estándar de codificación del *JPEG* es un conjunto de números entre *00* y *FF* en el sistema hexadecimal. Aunque como

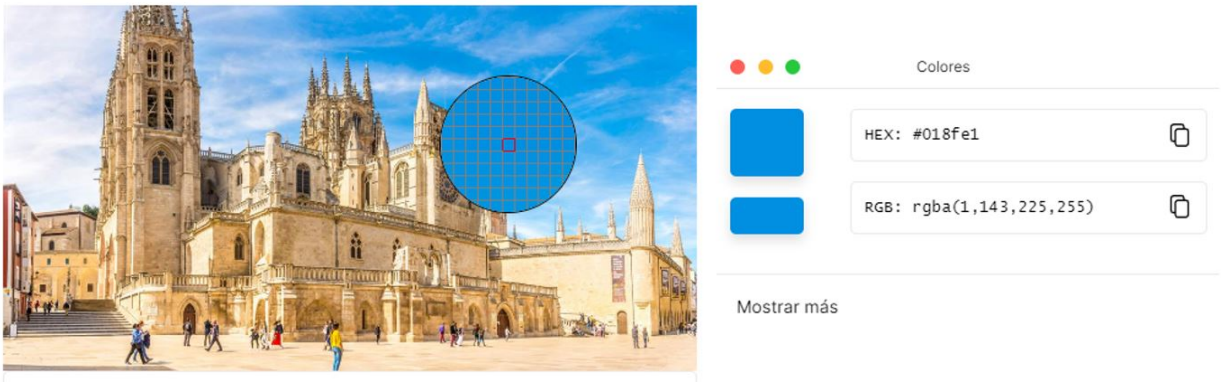


Figura 1.4: Representación de un píxel de la imagen de la Catedral de Burgos.

nos indica la leyenda, el código de la imagen se puede interpretar y está separado mediante marcadores y cabeceras en diferentes fragmentos, en este trabajo no se entrará a explicar esta codificación.

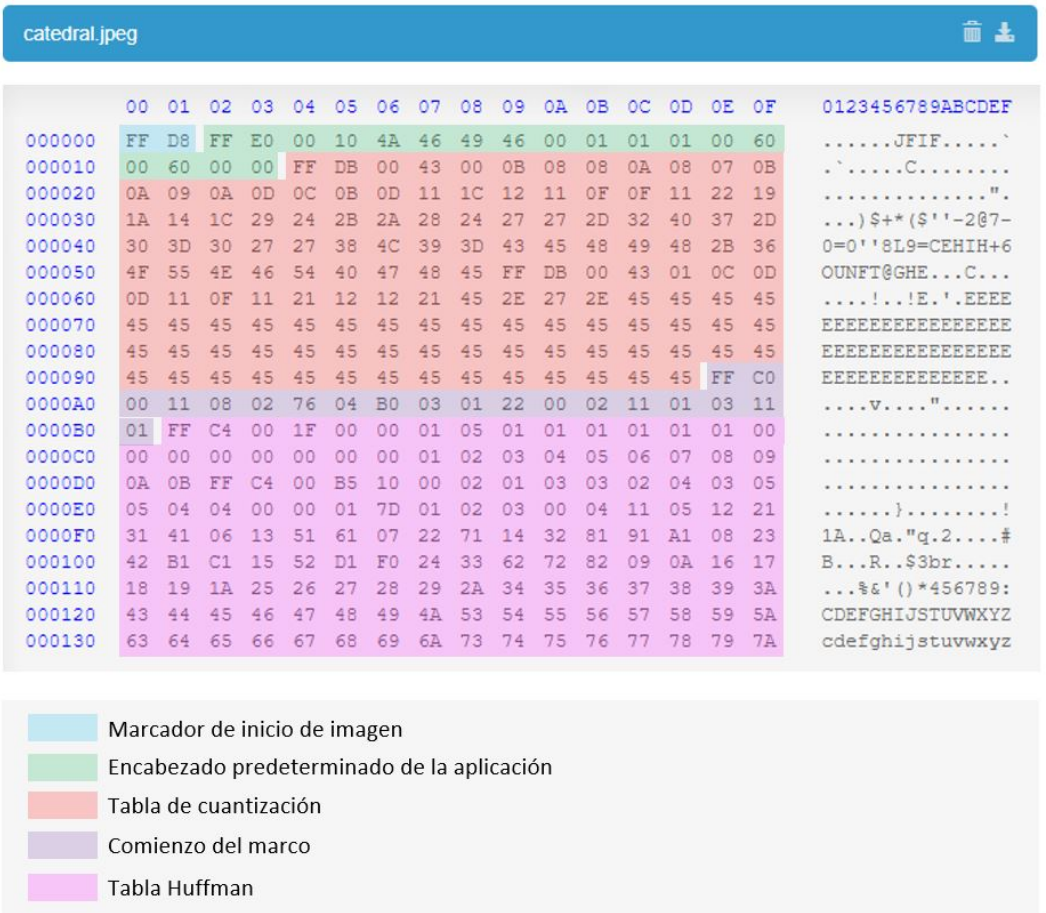


Figura 1.5: Codificación del *JPEG* de la imagen de la Catedral de Burgos. En general, la codificación del *JPEG* sigue el patrón: marcador (*FF*___); entero de 16 bits, que indica el numero de bits que deben leerse a la derecha; datos. Existen unas asignaciones para los diferentes marcadores. Por ejemplo, el marcador *FFD8* es el inicio de la imagen.

1.1. Procesamiento de señales

Una **señal** se produce por la variación de una o más magnitudes físicas (como la presión del aire o el voltaje de una corriente eléctrica) con respecto a una variable controlable (normalmente el tiempo o el espacio). El emisor envía la señal por algún medio y el receptor recibe esa señal y realiza su correspondiente interpretación.

Es frecuente modelizar una señal como una función matemática $f : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Si representamos $f = (f_1, \ldots, f_n)$, bastará con estudiar funciones real valuadas $f_i : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}$. Una primera clasificación de señales (ver Figura 1.6) viene dada en función de la naturaleza del dominio D.

- **Señales analógicas**: su dominio es continuo (un abierto de $\mathbb{R}^n$, por ejemplo).
- **Señales digitales**: su dominio es discreto.

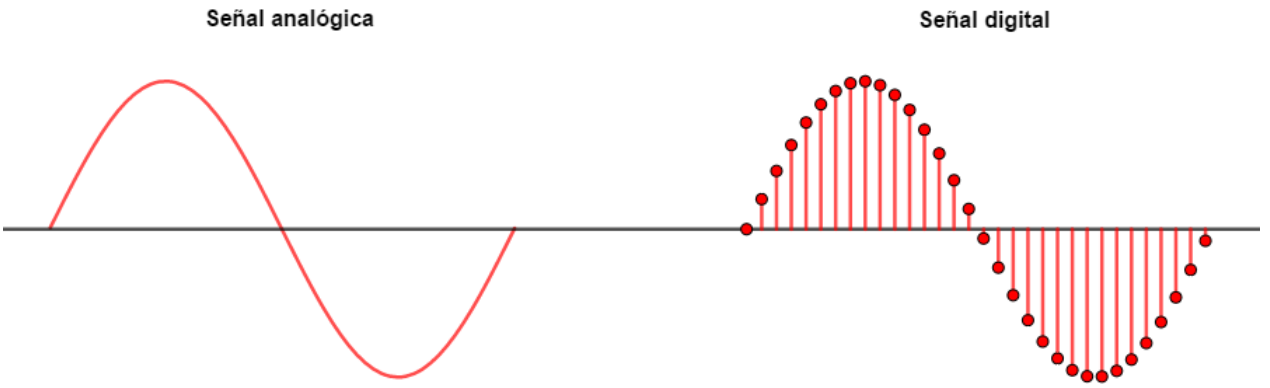


Figura 1.6: Esquemas gráficos de una señal analógica y de una digital.

Aunque las imágenes analógicas son de gran utilidad, lo cierto es que en las aplicaciones del mundo real debemos ceñirnos a dominios discretos. Por ejemplo, un ordenador no puede manejar infinitos datos. En este trabajo se considerarán de manera recurrente las imágenes digitales, que interpretaremos matemáticamente como señales bidimensionales digitales y cuyo conjunto de valores posibles será $\{0, 1, 2, \ldots, 255\}$ (imágenes monocromáticas) o $\{(0, 0, 0), (0, 1, 0), \ldots, (255, 255, 255)\}$ (color *RGB*).

El **procesamiento de señales** (*Signal Processing*) es el área científica que se encarga del estudio de las señales. Dentro de su ámbito encontramos el procesamiento de imágenes digitales, que se define como la agrupación de técnicas que realizamos sobre las imágenes digitales con el fin de mejorarlas, ya sea logrando que estas tengan unas características específicas u obteniendo cierta información particular de las mismas. En esta disciplina encontramos cuestiones como la restauración de imágenes, la mejora de calidad de las imágenes o la compresión y codificación de imágenes, técnicas que enmarcan el ámbito de trabajo del *JPEG*.

1.2. Técnicas de compresión de imágenes

Representar una imagen en un formato compacto no sirve de nada si no se hace de forma eficiente, es decir, no nos será útil reducir el tamaño que ocupa una imagen si estamos perdiendo información relevante en el proceso. A partir de esta idea se desarrollan las bases de la compresión de un fichero. Se trata de reducir la redundancia e irrelevancia de un archivo minimizando al mismo tiempo la pérdida de información. El objetivo de cualquier técnica de compresión de imágenes es representar imágenes con menos información de modo que se ahorren costes de almacenamiento, tiempo y transmisión.

1.2.1. Tipos de compresión

Las distintas técnicas de codificación y decodificación de imágenes se clasifican en dos tipos en función de la naturaleza de la compresión:

- **Compresión sin pérdida (*lossless*):** como su nombre indica, este tipo se fundamenta en condensar los datos de la imagen, pero sin desestimar ninguna información. Por ello, cuando recuperemos la fotografía después de haberla comprimido, esta será idéntica a la original. Tiene como ventaja principal que se mantiene absolutamente la calidad de la imagen, pero, por otro lado, posee el inconveniente de que se limita la capacidad de compresión, que será mucho menor que en un procedimiento de compresión con pérdida. Pertenecientes a este tipo, encontramos formatos como *PNG* (*Portable Network Graphics*) o *RLE* (*Run-Length Encoding*).

- **Compresión con pérdida (*lossy*):** a diferencia del anterior, no se almacena toda la información de la imagen, aunque esto conlleve la pérdida justificada de algunos datos de la imagen. Básicamente, se estima como asumible perder algo de calidad en la imagen a cambio de reducir de forma significativa el tamaño de la misma. Cuanto mayor sea el grado de compresión, peor será la aproximación en la recuperación de la imagen. Ejemplos de estos algoritmos son *JPEG 2000* o *GIF* (*Graphics Interchange Format*). En particular, el estándar del que trata este trabajo, *JPEG*, pertenece a este tipo de compresión.

1.2.2. Elementos de compresión

- **Formato de intercambio (*interchange format*):** se trata del formato específico utilizado para representar datos de imagen comprimidos con el fin de intercambiarlos entre diferentes aplicaciones. Para cumplir con los requisitos del formato, los datos de imagen comprimidos deben estar codificados de acuerdo con el proceso de decodificación seleccionado. Se especifican, además, una serie de pruebas para verificar si los datos comprimidos cumplen con estos requisitos.

Es decir, el formato de intercambio es una forma de compartir, entre diferentes programas o sistemas informáticos, datos de imagen comprimidos y estos deben estar codificados de cierta manera para que puedan ser decodificados correctamente.

- **Codificador (*encoder*):** es la herramienta que convierte los datos de imagen originales en datos de imagen comprimidos. Para ello, utiliza uno o más procesos de codificación especificados en las normas y especificaciones técnicas del estándar correspondiente. El objetivo del codificador es reducir el tamaño del archivo de imagen comprimida sin perder una cantidad significativa de información. Para lograrlo, un codificador debe ser capaz de convertir con precisión los datos de la imagen original en datos de imagen comprimidos que cumplan con la sintaxis del formato de intercambio especificado.

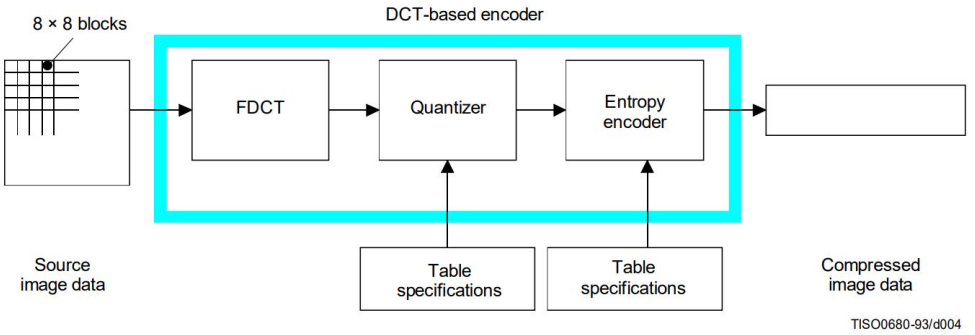


Figura 1.7: Diagrama simplificado de un codificador que utiliza la *DCT*.

- **Decodificador (*decoder*):** se utiliza para reconstruir los datos de imagen comprimidos para que se puedan ver o procesar en su formato original. A pesar de que mediante el proceso de compresión se consigue reducir el tamaño de los archivos de imagen para poder almacenarlos o transmitirlos de un manera más eficiente, es necesario descomprimirlos posteriormente para que estos datos sean útiles. Para ello, el decodificador debe seguir con la precisión adecuada los distintos procesos de decodificación especificados en el estándar correspondiente.

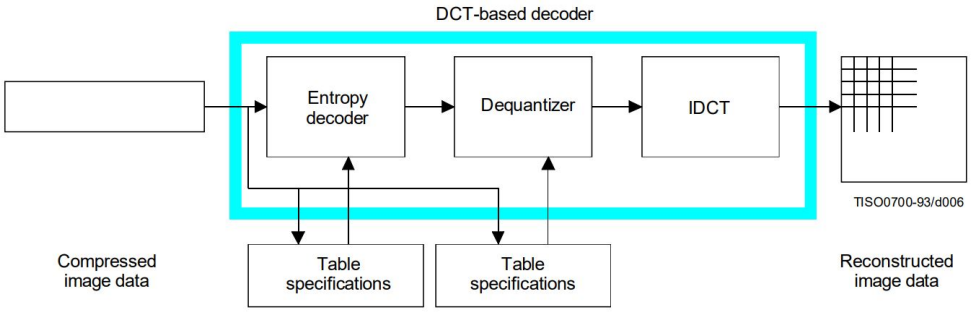


Figura 1.8: Diagrama simplificado de un decodificador que utiliza la *DCT*.

En las Figuras 1.7 y 1.8 se pueden ver ejemplos de esquemas de un *encoder* y un *decoder* [16] basados en la transformada discreta de coseno, que introduciremos en el capítulo 2.

1.3. Fundamentos del algoritmo del *JPEG*

El estándar de compresión y codificación de imágenes del *JPEG* considera varios procesos de codificación y decodificación. En este trabajo nos centraremos en el más simple y más habitual, el *baseline sequential process*, que se cataloga como *lossy* por estar basado en la transformada discreta de coseno. Encontramos cuatro fases para comprimir un archivo:

1. **Descomposición en bloques y tratamiento del color:** la imagen se separa en bloques llamados *MCU* (*Minimum Coded Units*) y se obtienen los valores de color *RGB* de cada uno de ellos. Además, se hacen las transformaciones necesarias para pasar estos términos a otro espacio de color (YC_bC_r) más apropiado.

2. **Transformada discreta de coseno:** se utiliza este objeto matemático para almacenar de forma eficiente los valores de cada componente de cada *MCU*.

3. **Proceso de cuantización:** se ejecuta un proceso de compresión con pérdida aplicando un proceso de cuantización a los coeficientes obtenidos en la fase anterior.

4. **Proceso de codificación:** mediante la aplicación de diferentes técnicas (zig-zag transversal, *Run-Length Encoding*, *Delta Encoding* y codificación Huffman) se realizan varias transformaciones a los coeficientes hasta llegar al archivo comprimido.

Para obtener la imagen original (o, mejor dicho, una parecida a la original) se aplican estas cuatro etapas en el orden inverso.

La organización de lo que resta de memoria es la siguiente: en el capítulo 2 se incluyen algunos conceptos básicos del análisis de Fourier, hasta presentar la transformada discreta de coseno y un algoritmo para calcularla de forma eficiente; en el capítulo 3 se describen de manera más precisa las cuatro etapas anteriormente descritas del proceso de compresión y codificación; en el penúltimo capítulo aparecen las conclusiones del trabajo. Se presenta una bibliografía final.

Capítulo 2

Conceptos elementales del análisis de Fourier

Jean-Baptiste Joseph Fourier (1768–1830) fue un matemático y científico francés, reconocido principalmente por resolver la ecuación diferencial del calor a través de las denominadas series de Fourier (método de Fourier o de separación de variables). Sus trabajos y avances fueron el inicio de la rama del análisis de Fourier en la que se estudia la representabilidad de una función por medio de estas series trigonométricas particulares.

Figura 2.1: Retrato de Joseph Fourier.

Sus contribuciones científicas fueron numerosas, destacando su estudio sobre la termodinámica así como la modelización matemática de fenómenos físicos. Estos análisis le llevaron a publicar, en 1822, un tratado titulado *Théorie analytique de la chaleur*. En él, Fourier deduce y resuelve la ecuación diferencial que describía la difusión del calor en los cuerpos sólidos, hoy en día conocida como la ley de conducción térmica de Fourier. La resolución se basa en el uso de series infinitas de funciones trigonométricas (senos y cosenos): las series de Fourier. Esta herramienta matemática, que posteriormente daría paso a la transformada

de Fourier, fue fundamental en diversas áreas de ingeniería y física. Asimismo, su utilidad ha sido imprescindible en multitud de campos entre los que se encuentran las matemáticas abstractas, el análisis vibratorio o el procesamiento de señales e imágenes y, en particular, en la compresión de datos.

Fourier nació en Auxerre en 1768. Hijo de un sastre, su infancia estuvo marcada por las dificultades económicas, así como por el fallecimiento de sus padres cuando él tenía ocho años, y de seis de sus hermanos. Estudió en la Escuela Militar de su ciudad hasta los 14 años, donde sobresalió desde el principio en todas las materias y, en especial, en las matemáticas. Desde 1794 fue alumno de ilustres profesores, como Pierre-Simon Laplace y Joseph-Louis Lagrange, en la Escuela Normal Superior de París donde tiempo ocuparía el puesto dejado por el último en la cátedra de Análisis y Mecánica.

A Fourier le tocó vivir la Revolución Francesa y se sabe que participó en la Campaña de Egipto[1] en 1798 que organizó el general Napoleón Bonaparte. Sus logros en el área de las ciencias le llevarían a fundar, en 1810, la Facultad Imperial de Grenoble y, 5 años después, la Academia del Delfinado. En 1817, Fourier pasó a formar parte de la prestigiosa Academia de Ciencias de Francia, y tiempo después, en 1822, fue nombrado como secretario perpetuo de las secciones de matemáticas y física.

2.1. Series de Fourier

El problema de representar una función 2π-periódica $f(x)$ mediante una serie trigonométrica de la forma

$$a_0 + 2\sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx)\right)$$

surge de manera natural en varios problemas de la física matemática. Si, en efecto,

$$f(x) = a_0 + 2\sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx)\right), \qquad 0 \leq x \leq 2\pi, \tag{2.1}$$

entonces los coeficientes a_n y b_n vienen dados por las expresiones

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(nx)\ dx, \qquad n \geq 0 \tag{2.2}$$

y

$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(nx)\ dx, \qquad n \geq 1. \tag{2.3}$$

[1]La Campaña francesa de Egipto no fue únicamente una campaña militar. Muchos intelectuales franceses se desplazaron a Egipto para estudiar la historia antigua del país. Ejemplo de ello es el descubrimiento de la piedra Rosetta, que permitió descifrar los jeroglíficos egipcios.

Para ver esto, basta con recordar las relaciones de ortogonalidad de las funciones trigonométricas seno y coseno. Se tiene que

$$\frac{1}{2\pi}\int_0^{2\pi}\cos(mx)\cos(nx)\,dx=\begin{cases}0, & \text{si} \quad m\neq n,\ m\geq 0,\ n\geq 0,\\ \frac{1}{2}, & \text{si} \quad m=n,\ m>0,\ n>0,\\ 1, & \text{si} \quad m=n=0,\end{cases} \tag{2.4}$$

$$\frac{1}{2\pi}\int_0^{2\pi}\sin(mx)\sin(nx)\,dx=\begin{cases}0, & \text{si} \quad m\neq n,\ m\geq 0,\ n\geq 0,\\ \frac{1}{2}, & \text{si} \quad m=n,\ m>0,\ n>0,\\ 0, & \text{si} \quad m=n=0,\end{cases} \tag{2.5}$$

$$\frac{1}{2\pi}\int_0^{2\pi}\cos(mx)\sin(nx)\,dx=\ 0, \quad \text{si} \quad m\geq 0,\ n\geq 0. \tag{2.6}$$

Bajo ciertas condiciones (por ejemplo, si f es integrable Lebesgue en $[0,2\pi]$ y la serie converge uniformemente), podemos multiplicar por $\cos(mx)$ a ambos lados de (2.1) e integrar término a término. Utilizando las ecuaciones (2.4) y (2.6) se obtiene la expresión (2.2). Si en lugar de multiplicar por $\cos(mx)$ a ambos lados de (2.1), multiplicamos por $\sin(mx)$ y utilizamos las ecuaciones (2.5) y (2.6), obtenemos la expresión (2.3).

Dada una función 2π-periódica $f(x)$ definida e integrable Lebesgue en $[0,2\pi]$, denominamos serie de Fourier de f a la serie trigonométrica

$$a_0+2\sum_{n=1}^{\infty}\left(a_n\cos(nx)+b_n\sin(nx)\right),$$

donde los coeficientes a_n y b_n se llaman coeficientes de Fourier y se definen como en las ecuaciones (2.2) y (2.3). Aunque hasta ahora solamente hemos hablado de series de Fourier de funciones definidas en el intervalo $[0,2\pi]$, podemos extender el mismo razonamiento a un intervalo cualquiera $[0,2L]$ si asignamos $t=\frac{Lx}{\pi}$ en la serie de Fourier.

Dada una función f definida e integrable Lebesgue en $[0,2L]$ y $2L$-periódica, denominamos **serie de Fourier** [10] de f a la expresión

$$a_0+2\sum_{n=1}^{\infty}\left(a_n\cos\left(\frac{\pi nt}{L}\right)+b_n\sin\left(\frac{\pi nt}{L}\right)\right) \tag{2.7}$$

donde los **coeficientes de Fourier** a_n y b_n se definen ahora como

$$a_n=\frac{1}{L}\int_0^{2L}f(t)\cos\left(\frac{\pi nt}{L}\right)dt, \qquad n\geq 0,$$

y

$$b_n=\frac{1}{L}\int_0^{2L}f(t)\sin\left(\frac{\pi nt}{L}\right)dt, \qquad n\geq 1.$$

Para definir la transformada discreta de Fourier vamos a partir de otra forma de la ecuación (2.7) vista anteriormente. Sabemos que dada una función periódica $f(t)$ de periodo L, en general, se cumple que

$$f(t) = a_0 + 2\sum_{n=1}^{\infty}\left(a_n \cos\left(\frac{2\pi nt}{L}\right) + b_n \sin\left(\frac{2\pi nt}{L}\right)\right), \tag{2.8}$$

donde definimos los coeficientes a_n y b_n como

$$a_n = \frac{1}{L}\int_0^L f(t)\cos\left(\frac{2\pi nt}{L}\right)\,dt, \qquad n \geq 0,$$
$$b_n = \frac{1}{L}\int_0^L f(t)\sin\left(\frac{2\pi nt}{L}\right)\,dt, \qquad n \geq 1.$$

Podemos combinar las dos ecuaciones anteriores definiendo

$$c_n = a_n - ib_n, \qquad n \geq 0$$

(asumimos que $b_0 = 0$). Usando la fórmula de Euler

$$e^{-i\left(\frac{2\pi nt}{L}\right)} = \cos\left(\frac{2\pi nt}{L}\right) - i\sin\left(\frac{2\pi nt}{L}\right),$$

resulta que

$$c_n = \frac{1}{L}\int_0^L f(t)e^{-i\left(\frac{2\pi nt}{L}\right)}\,dt, \qquad n \geq 0. \tag{2.9}$$

2.2. Transformada discreta de Fourier

En la práctica, los ordenadores solo pueden manejar una cantidad finita de información. Esto implica que, a la postre, se manejarán funciones digitales en vez de analógicas. Así, una señal analógica L-periódica se muestrea en N puntos equiespaciados por un intervalo de muestreo constante $\Delta = \frac{L}{N}$, para dar una señal discreta que vendrá representada por una sucesión discreta $\{x_r\}_{r\in\mathbb{Z}_N}$, donde $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$. De modo que x_r será el valor de $x(t)$ en el instante $t = r\Delta$ y aproximaremos c_n en la ecuación (2.9) mediante la fórmula

$$X_n = \frac{1}{N\Delta}\sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr\Delta}{N\Delta}\right)}\Delta, \qquad n \in \mathbb{Z}_N$$

que, simplificando, es la transformada discreta de Fourier o *DFT* (*Discrete Fourier Transform*), una discretización de (2.9) [13].

Luego, dada una sucesión discreta $\{x_r\}_{r\in\mathbb{Z}_N}$, definimos la **transformada discreta de Fourier** de $\{x_r\}_{r\in\mathbb{Z}_N}$ por

$$X_n = \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{N}\right)}, \qquad n \in \mathbb{Z}_N. \tag{2.10}$$

Hay que decir que, en general, denotaremos a los elementos de la sucesión de la *DFT* con la letra mayúscula de la letra que se use para la sucesión de la que estamos calculando su transformada, como hemos hecho aquí.

A partir de los valores X_n, es posible recuperar la sucesión $\{x_r\}_{r\in\mathbb{Z}_N}$ con exactitud si se utiliza el operador inverso de la *DFT*. Dada una sucesión $\{G_n\}_{n\in\mathbb{Z}_N}$, denominamos **transformada discreta de Fourier inversa** o *IDFT* a

$$\sum_{n=0}^{N-1} G_n e^{i\left(\frac{2\pi nr}{N}\right)}, \qquad r \in \mathbb{Z}_N.$$

Como ya hemos adelantado,

$$x_r = \sum_{n=0}^{N-1} X_n e^{i\left(\frac{2\pi nr}{N}\right)}. \tag{2.11}$$

En efecto,

$$\sum_{n=0}^{N-1} X_n e^{i\left(\frac{2\pi nr}{N}\right)} = \sum_{n=0}^{N-1} \left(\frac{1}{N} \sum_{k=0}^{N-1} x_k e^{-i\left(\frac{2\pi nk}{N}\right)} \right) e^{i\left(\frac{2\pi nr}{N}\right)} = \frac{1}{N} \sum_{k=0}^{N-1} x_k \sum_{n=0}^{N-1} e^{i\left(\frac{2\pi n(r-k)}{N}\right)}.$$

Se tiene que

$$\sum_{n=0}^{N-1} e^{i\left(\frac{2\pi n(r-k)}{N}\right)} = \begin{cases} 0, & \text{si} \quad r \neq k, \\ \\ N, & \text{si} \quad r = k. \end{cases}$$

Luego,

$$\frac{1}{N} \sum_{k=0}^{N-1} x_k \sum_{n=0}^{N-1} e^{i\left(\frac{2\pi n(r-k)}{N}\right)} = x_r,$$

y la expresión (2.11) queda demostrada.

2.3. Algoritmo de la transformada rápida de Fourier

El proceso para calcular computacionalmente la transformada discreta de Fourier puede llegar a ser muy costoso. De hecho, tal y como hemos visto antes, con la *DFT* tratamos de obtener a partir de otra sucesión $\{x_r\}_{r\in\mathbb{Z}_N}$, la sucesión $\{X_n\}_{n\in\mathbb{Z}_N}$ que está definida por

$$X_n = \frac{1}{N} \sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{N}\right)}, \qquad n \in \mathbb{Z}_N. \tag{2.12}$$

Como estamos realizando una suma de N términos, donde la n puede tomar N valores, calcular $\{X_n\}$ de forma directa en realidad implica una operación con un coste computacional de orden N^2 (en el peor de los casos). Si N es un número muy grande (como suele ser el caso en la práctica), este proceso puede llevar demasiado tiempo para un ordenador. Por ello, con el objetivo de aumentar la eficiencia del cálculo, aparece la necesidad de utilizar un algoritmo que reduzca considerablemente el número de operaciones.

La **transformada rápida de Fourier** o *FFT* (*Fast Fourier Transform*) es un algoritmo que simplifica el coste computacional de la *DFT*. Este algoritmo, que impulsó de forma significativa muchas aplicaciones en el procesamiento de señales, fue descrito en [8, 1965] por los matemáticos estadounidenses James William Cooley (1926–2016) y John Wilder Tukey (1915–2000).

El método de Cooley y Tukey, a diferencia de otros algoritmos de duplicación sucesiva creados por matemáticos como P. Rudnick, es más general, ya que no tiene la restricción de ser solo aplicable cuando N es una potencia de 2 (ver [7]). De este modo, si podemos descomponer N en dos factores de tal forma que $N = p \cdot q$ mediante una transformación bidimensional tenemos como resultado un algoritmo con un coste computacional de orden $N(p+q)$ en vez de N^2. De hecho, si N es una potencia de 2 esto se traduce en un coste de orden $N \log_2 N$.

El método es un buen ejemplo del conocido principio "divide y vencerás": en vez de calcular directamente la *DFT* de la sucesión original, se divide esta en sucesiones más pequeñas y se aplica la *DFT* a cada una de ellas. Después, al combinar de cierta forma los resultados obtenidos, se consigue la *DFT* completa deseada. Veamos esto con más detalle a continuación.

Sea N un número par y $\{X_n\}_{n\in\mathbb{Z}_N}$ la *DFT* de la sucesión $\{x_r\}_{r\in\mathbb{Z}_N}$. Dividamos esta última en dos subsucesiones $\{y_r\}_{r=0}^{\frac{N}{2}-1}$ y $\{z_r\}_{r=0}^{\frac{N}{2}-1}$, donde la primera va a estar compuesta por los elementos de $\{x_r\}$ de las posiciones pares y la segunda por los elementos de $\{x_r\}$ de las posiciones impares, es decir,

$$\left.\begin{aligned} y_r &= x_{2r} \\ z_r &= x_{2r+1} \end{aligned}\right\} \; r \in \mathbb{Z}_{N/2}.$$

Notemos que las *DFT* de estas dos subsucesiones son (ver (2.10))

$$\left.\begin{aligned} Y_n &= \frac{1}{N/2}\sum_{r=0}^{N/2-1} y_r e^{-i\left(\frac{2\pi nr}{N/2}\right)} \\ Z_n &= \frac{1}{N/2}\sum_{r=0}^{N/2-1} z_r e^{-i\left(\frac{2\pi nr}{N/2}\right)} \end{aligned}\right\} \; n \in \mathbb{Z}_{N/2}.$$

Así, podemos reescribir la expresión de la *DFT* original como

$$\begin{aligned} X_n &= \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{N}\right)} \\ &= \frac{1}{N}\left(\sum_{r=0}^{N/2-1} x_{2r} e^{-i\left(\frac{2\pi n(2r)}{N}\right)} + \sum_{r=0}^{N/2-1} x_{2r+1} e^{-i\left(\frac{2\pi n(2r+1)}{N}\right)}\right) \\ &= \frac{1}{2}\left(\frac{1}{N/2}\sum_{r=0}^{N/2-1} y_r e^{-i\left(\frac{2\pi nr}{N/2}\right)} + \frac{e^{-i\left(\frac{2\pi n}{N}\right)}}{N/2}\sum_{r=0}^{N/2-1} z_r e^{-i\left(\frac{2\pi nr}{N/2}\right)}\right) \\ &= \frac{1}{2}\left(Y_n + e^{-i\left(\frac{2\pi n}{N}\right)} Z_n\right). \end{aligned} \tag{2.13}$$

La ventaja de la fórmula (2.13) es que expresa la *DFT* de la sucesión $\{x_r\}_{r\in\mathbb{Z}_N}$ como una suma de las *DFT* de las sucesiones más cortas. Sin embargo, debemos tener en cuenta que esta descomposición únicamente es válida en el rango $n \in \{0, 1, 2, \ldots, \frac{N}{2} - 1\}$, mientras que la definición de los valores X_n es para todo $n \in \{0, 1, 2, \ldots, N-1\}$. Para obtener los valores de X_n en el rango $n \in \{\frac{N}{2}, \frac{N}{2} + 1, \ldots, N-1\}$, procedemos del siguiente modo:

$$\begin{aligned} X_n &= \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{N}\right)} = \frac{1}{N}\sum_{r=0}^{N/2-1} x_{2r} e^{-i\left(\frac{2\pi n(2r)}{N}\right)} + \frac{1}{N}\sum_{r=0}^{N/2-1} x_{2r+1} e^{-i\left(\frac{2\pi n(2r+1)}{N}\right)} \\ &= \frac{1}{2}\left(Y_{n-\frac{N}{2}} + e^{-i\left(\frac{2\pi n}{N}\right)} Z_{n-\frac{N}{2}}\right). \end{aligned}$$

Esto es equivalente a poner

$$X_{k+\frac{N}{2}} = \frac{1}{2}\left(Y_k + e^{-i\left(\frac{2\pi(k+N/2)}{N}\right)} Z_k\right) = \frac{1}{2}\left(Y_k + e^{-i\left(\frac{2\pi k}{N}+\pi\right)} Z_k\right), \qquad k \in \mathbb{Z}_{N/2},$$

y como $e^{-i\pi} = -1$, llegamos a que

$$X_{k+\frac{N}{2}} = \frac{1}{2}\left(Y_k - e^{-i\left(\frac{2\pi k}{N}\right)} Z_k\right), \qquad k \in \mathbb{Z}_{N/2}.$$

Por lo tanto,

$$\left.\begin{aligned} X_n &= \frac{1}{2}\left(Y_n + e^{-i\left(\frac{2\pi n}{N}\right)} Z_n\right) \\ X_{n+\frac{N}{2}} &= \frac{1}{2}\left(Y_n - e^{-i\left(\frac{2\pi n}{N}\right)} Z_n\right) \end{aligned}\right\} \; n \in \mathbb{Z}_{N/2}. \tag{2.14}$$

La esencia del razonamiento anterior es que se puede obtener la *DFT* de una sucesión inicial $\{x_r\}_{r\in\mathbb{Z}_N}$ a partir del cálculo de las *DFT* de dos subsucesiones con menos términos. De hecho, si la sucesión original tiene una potencia de 2 como número de elementos, podemos repetir el mismo proceso con las subsucesiones hasta llegar a un nivel donde las subsucesiones tengan únicamente un solo término. Como vemos a continuación, el cálculo de las *DTF* de estas últimas es inmediato.

Por la ecuación (2.10), sabemos que dada una sucesión $\{x_r\}_{r\in\mathbb{Z}_N}$, su *DFT* es otra sucesión $\{X_n\}_{n\in\mathbb{Z}_N}$ tal que

$$X_n = \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{N}\right)}.$$

Si $N = 1$, se cumple que $X_0 = x_0$. Por tanto, la *DFT* de una sucesión que tiene un solo término es ese mismo término.

A continuación, aplicamos este algoritmo en un ejemplo sencillo, donde la sucesión inicial no tiene muchos términos. Así, supongamos que $N = 8$ y que tenemos una sucesión finita $\{x_r\}_{r=0}^{7}$ a la que vamos a aplicar la *DFT*. Por la fórmula (2.10),

$$X_n = \frac{1}{8}\sum_{r=0}^{8-1} x_r e^{-i\left(\frac{2\pi nr}{8}\right)} = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{\pi nr}{4}\right)}, \qquad n \in \mathbb{Z}_8.$$

Obtenemos, por tanto, la sucesión $\{X_n\}_{n=0}^{7}$ con los términos definidos por

$$X_0 = \frac{1}{8}\sum_{r=0}^{7} x_r = \frac{1}{8}\left(x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7\right),$$

$$X_1 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{\pi r}{4}\right)} = \frac{1}{8}\left(x_0 + e^{-\frac{i\pi}{4}}x_1 - ix_2 + -e^{\frac{i\pi}{4}}x_3 - x_4 - e^{-\frac{i\pi}{4}}x_5 + ix_6 + e^{\frac{i\pi}{4}}x_7\right),$$

$$X_2 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{\pi r}{2}\right)} = \frac{1}{8}\left(x_0 - ix_1 - x_2 + ix_3 + x_4 - ix_5 - x_6 + ix_7\right),$$

$$X_3 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{3\pi r}{4}\right)} = \frac{1}{8}\left(x_0 - e^{\frac{i\pi}{4}}x_1 + ix_2 + e^{-\frac{i\pi}{4}}x_3 - x_4 + e^{\frac{i\pi}{4}}x_5 - ix_6 - e^{-\frac{i\pi}{4}}x_7\right),$$

$$X_4 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i(\pi r)} = \frac{1}{8}\left(x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 - x_7\right),$$

$$X_5 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{5\pi r}{4}\right)} = \frac{1}{8}\left(x_0 - e^{-\frac{i\pi}{4}}x_1 - ix_2 + e^{\frac{i\pi}{4}}x_3 - x_4 + e^{-\frac{i\pi}{4}}x_5 + ix_6 - e^{\frac{i\pi}{4}}x_7\right),$$

$$X_6 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{3\pi r}{2}\right)} = \frac{1}{8}\left(x_0 + ix_1 - x_2 - ix_3 + x_4 + ix_5 - x_6 - ix_7\right),$$

$$X_7 = \frac{1}{8}\sum_{r=0}^{7} x_r e^{-i\left(\frac{7\pi r}{4}\right)} = \frac{1}{8}\left(x_0 + e^{\frac{i\pi}{4}}x_1 + ix_2 - e^{-\frac{i\pi}{4}}x_3 - x_4 - e^{\frac{i\pi}{4}}x_5 - ix_6 + e^{-\frac{i\pi}{4}}x_7\right).$$

Si en vez de proceder de manera directa, aplicamos el algoritmo de la *FFT*, debemos empezar por dividir la sucesión inicial $\{x_r\}_{r=0}^{7}$ en subsucesiones, y como N es una potencia de 2, continuamos así hasta obtener sucesiones de un solo término. En estas, sabemos que sus *DFT* son ellas mismas, así que iremos combinando con (2.14) los resultados obtenidos hasta obtener la *DFT* de la sucesión inicial. Este proceso queda reflejado en la Figura 2.2.

Como hemos explicado anteriormente, la *DFT* de las subsucesiones de un solo término se calcula trivialmente. Con la notación utilizada en dicha figura,

$$\{A_n\} = x_0, \qquad \{B_n\} = x_4, \qquad \{C_n\} = x_2, \qquad \{D_n\} = x_6,$$
$$\{E_n\} = x_1, \qquad \{F_n\} = x_5, \qquad \{G_n\} = x_3, \qquad \{H_n\} = x_7.$$

A continuación, mediante la ecuación (2.14) combinamos las sucesiones que hemos obtenido hasta llegar a $\{X_n\}_{n=0}^{7}$:

$$V_0 = \frac{1}{2}\left(A_0 + e^{-i\left(\frac{2\pi 0}{2}\right)}B_0\right) = \frac{1}{2}\left(x_0 + x_4\right),$$
$$V_1 = \frac{1}{2}\left(A_0 - e^{-i\left(\frac{2\pi 0}{2}\right)}B_0\right) = \frac{1}{2}\left(x_0 - x_4\right),$$
$$W_0 = \frac{1}{2}\left(C_0 + e^{-i\left(\frac{2\pi 0}{2}\right)}D_0\right) = \frac{1}{2}\left(x_2 + x_6\right),$$
$$W_1 = \frac{1}{2}\left(C_0 - e^{-i\left(\frac{2\pi 0}{2}\right)}D_0\right) = \frac{1}{2}\left(x_2 - x_6\right),$$
$$Y_0 = \frac{1}{2}\left(E_0 + e^{-i\left(\frac{2\pi 0}{2}\right)}F_0\right) = \frac{1}{2}\left(x_1 + x_5\right),$$

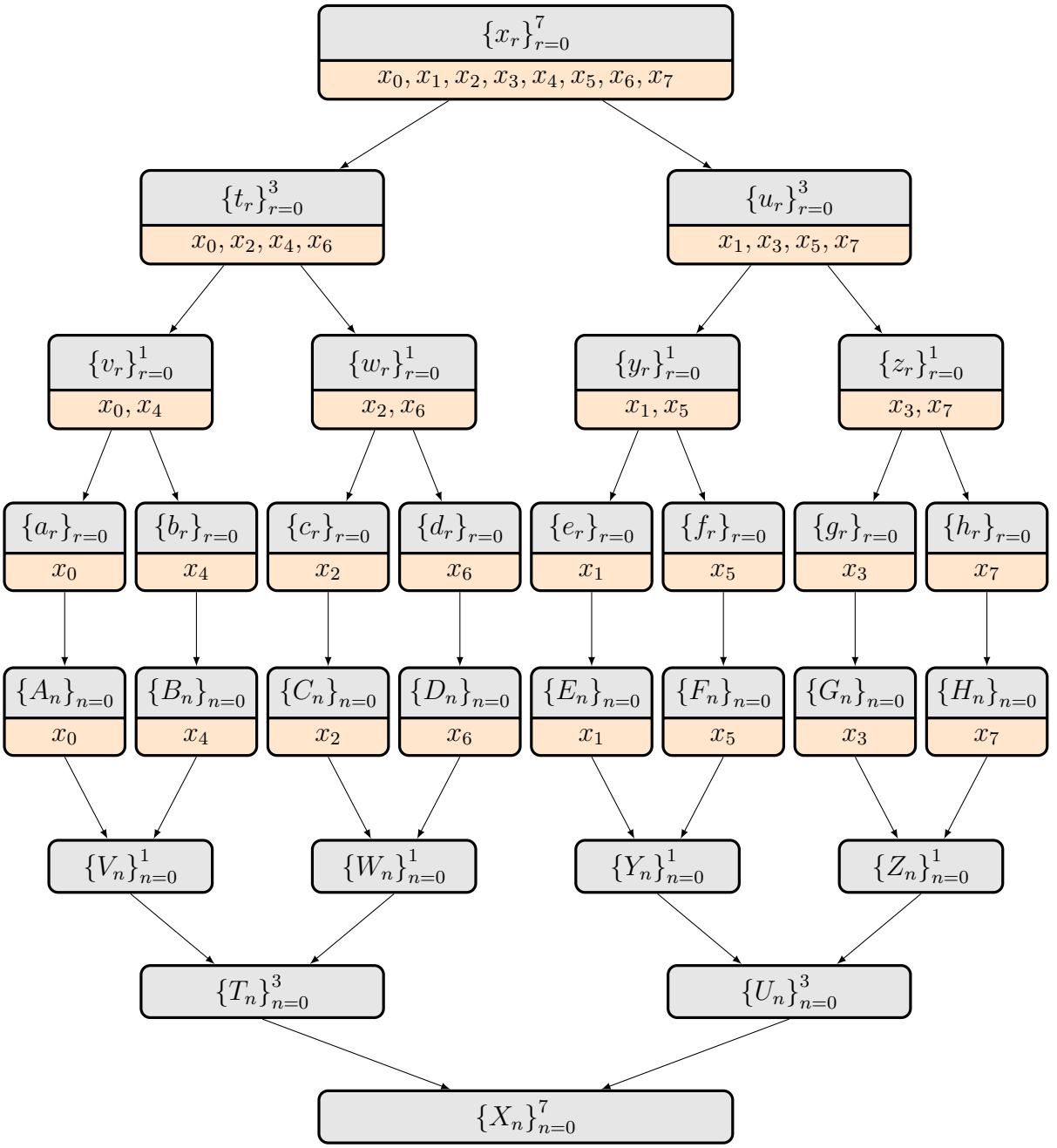


Figura 2.2: Diagrama de división y combinación del algoritmo de la *FFT*.

$$Y_1 = \frac{1}{2}\left(E_0 - e^{-i\left(\frac{2\pi 0}{2}\right)}F_0\right) = \frac{1}{2}\left(x_1 - x_5\right),$$
$$Z_0 = \frac{1}{2}\left(G_0 + e^{-i\left(\frac{2\pi 0}{2}\right)}H_0\right) = \frac{1}{2}\left(x_3 + x_7\right),$$
$$Z_1 = \frac{1}{2}\left(G_0 - e^{-i\left(\frac{2\pi 0}{2}\right)}H_0\right) = \frac{1}{2}\left(x_3 - x_7\right).$$

Ahora que hemos obtenido las sucesiones de orden 2, las combinamos de forma análoga para obtener $\{T_n\}_{n=0}^{3}$ y $\{U_n\}_{n=0}^{3}$.

$$T_0 = \frac{1}{2}\left(V_0 + e^{-i\left(\frac{2\pi 0}{4}\right)}W_0\right) = \frac{1}{4}\left(x_0 + x_2 + x_4 + x_6\right),$$

$$
\begin{aligned}
T_1 &= \frac{1}{2}\left(V_1 + e^{-i\left(\frac{2\pi 1}{4}\right)}W_1\right) = \frac{1}{4}\left(x_0 - ix_2 - x_4 + ix_6\right),\\
T_2 &= \frac{1}{2}\left(V_0 - e^{-i\left(\frac{2\pi 0}{4}\right)}W_0\right) = \frac{1}{4}\left(x_0 - x_2 + x_4 - x_6\right),\\
T_3 &= \frac{1}{2}\left(V_1 - e^{-i\left(\frac{2\pi 1}{4}\right)}W_1\right) = \frac{1}{4}\left(x_0 + ix_2 - x_4 - ix_6\right),\\
U_0 &= \frac{1}{2}\left(Y_0 + e^{-i\left(\frac{2\pi 0}{4}\right)}Z_0\right) = \frac{1}{4}\left(x_1 + x_3 + x_5 + x_7\right),\\
U_1 &= \frac{1}{2}\left(Y_1 + e^{-i\left(\frac{2\pi 1}{4}\right)}Z_1\right) = \frac{1}{4}\left(x_1 - ix_3 - x_5 + ix_7\right),\\
U_2 &= \frac{1}{2}\left(Y_0 - e^{-i\left(\frac{2\pi 0}{4}\right)}Z_0\right) = \frac{1}{4}\left(x_1 - x_3 + x_5 - x_7\right),\\
U_3 &= \frac{1}{2}\left(Y_1 - e^{-i\left(\frac{2\pi 1}{4}\right)}Z_1\right) = \frac{1}{4}\left(x_1 + ix_3 - x_5 - ix_7\right).
\end{aligned}
$$

Por último, combinamos $\{T_n\}_{n=0}^{3}$ y $\{U_n\}_{n=0}^{3}$ para formar la sucesión $\{X_n\}_{n=0}^{7}$, que se corresponde con la *DFT* de la sucesión original $\{x_r\}_{r=0}^{7}$

$$
\begin{aligned}
X_0 &= \frac{1}{2}\left(T_0 + e^{-i\left(\frac{2\pi 0}{8}\right)}U_0\right)\\
&= \frac{1}{8}\left(x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7\right),\\
X_1 &= \frac{1}{2}\left(T_1 + e^{-i\left(\frac{2\pi 1}{8}\right)}U_1\right)\\
&= \frac{1}{8}\left(x_0 + e^{-\frac{i\pi}{4}}x_1 - ix_2 - ie^{-\frac{i\pi}{4}}x_3 - x_4 - e^{-\frac{i\pi}{4}}x_5 + ix_6 + ie^{-\frac{i\pi}{4}}x_7\right),\\
X_2 &= \frac{1}{2}\left(T_2 + e^{-i\left(\frac{2\pi 2}{8}\right)}U_2\right)\\
&= \frac{1}{8}\left(x_0 - ix_1 - x_2 + ix_3 + x_4 - ix_5 - x_6 + ix_7\right),\\
X_3 &= \frac{1}{2}\left(T_3 + e^{-i\left(\frac{2\pi 3}{8}\right)}U_3\right)\\
&= \frac{1}{8}\left(x_0 + e^{-\frac{3i\pi}{4}}x_1 + ix_2 + ie^{-\frac{3i\pi}{4}}x_3 - x_4 - e^{-\frac{3i\pi}{4}}x_5 - ix_6 - ie^{-\frac{3i\pi}{4}}x_7\right),\\
X_4 &= \frac{1}{2}\left(T_0 - e^{-i\left(\frac{2\pi 0}{8}\right)}U_0\right)\\
&= \frac{1}{8}\left(x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 - x_7\right),\\
X_5 &= \frac{1}{2}\left(T_1 - e^{-i\left(\frac{2\pi 1}{8}\right)}U_1\right)\\
&= \frac{1}{8}\left(x_0 - e^{-\frac{i\pi}{4}}x_1 - ix_2 + ie^{-\frac{i\pi}{4}}x_3 - x_4 + e^{-\frac{i\pi}{4}}x_5 + ix_6 - ie^{-\frac{i\pi}{4}}x_7\right),\\
X_6 &= \frac{1}{2}\left(T_2 - e^{-i\left(\frac{2\pi 2}{8}\right)}U_2\right)\\
&= \frac{1}{8}\left(x_0 + ix_1 - x_2 - ix_3 + x_4 + ix_5 - x_6 - ix_7\right),\\
X_7 &= \frac{1}{2}\left(T_3 - e^{-i\left(\frac{2\pi 3}{8}\right)}U_3\right)\\
&= \frac{1}{8}\left(x_0 - e^{-\frac{3i\pi}{4}}x_1 + ix_2 - ie^{-\frac{3i\pi}{4}}x_3 - x_4 + e^{-\frac{3i\pi}{4}}x_5 - ix_6 + ie^{-\frac{3i\pi}{4}}x_7\right).
\end{aligned}
$$

Podemos simplificar estos resultados si tenemos en cuenta que

$$ie^{-\frac{i\pi}{4}} = e^{\frac{2i\pi}{4}}e^{-\frac{i\pi}{4}} = e^{\frac{i\pi}{4}} = -e^{-\frac{3i\pi}{4}} \qquad \text{y} \qquad -ie^{-\frac{i\pi}{4}} = e^{-\frac{2i\pi}{4}}e^{-\frac{i\pi}{4}} = e^{-\frac{3i\pi}{4}} = -e^{\frac{i\pi}{4}},$$

obteniendo

$$\begin{aligned}
X_0 &= \frac{1}{8}\left(x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7\right),\\
X_1 &= \frac{1}{8}\left(x_0 + e^{-\frac{i\pi}{4}}x_1 - ix_2 + -e^{\frac{i\pi}{4}}x_3 - x_4 - e^{-\frac{i\pi}{4}}x_5 + ix_6 + e^{\frac{i\pi}{4}}x_7\right),\\
X_2 &= \frac{1}{8}\left(x_0 - ix_1 - x_2 + ix_3 + x_4 - ix_5 - x_6 + ix_7\right),\\
X_3 &= \frac{1}{8}\left(x_0 - e^{\frac{i\pi}{4}}x_1 + ix_2 + e^{-\frac{i\pi}{4}}x_3 - x_4 + e^{\frac{i\pi}{4}}x_5 - ix_6 - e^{-\frac{i\pi}{4}}x_7\right),\\
X_4 &= \frac{1}{8}\left(x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 - x_7\right),\\
X_5 &= \frac{1}{8}\left(x_0 - e^{-\frac{i\pi}{4}}x_1 - ix_2 + e^{\frac{i\pi}{4}}x_3 - x_4 + e^{-\frac{i\pi}{4}}x_5 + ix_6 - e^{\frac{i\pi}{4}}x_7\right),\\
X_6 &= \frac{1}{8}\left(x_0 + ix_1 - x_2 - ix_3 + x_4 + ix_5 - x_6 - ix_7\right),\\
X_7 &= \frac{1}{8}\left(x_0 + e^{\frac{i\pi}{4}}x_1 + ix_2 - e^{-\frac{i\pi}{4}}x_3 - x_4 - e^{\frac{i\pi}{4}}x_5 - ix_6 + e^{-\frac{i\pi}{4}}x_7\right).
\end{aligned}$$

que coinciden con los que obtuvimos al aplicar la *DFT* directamente.

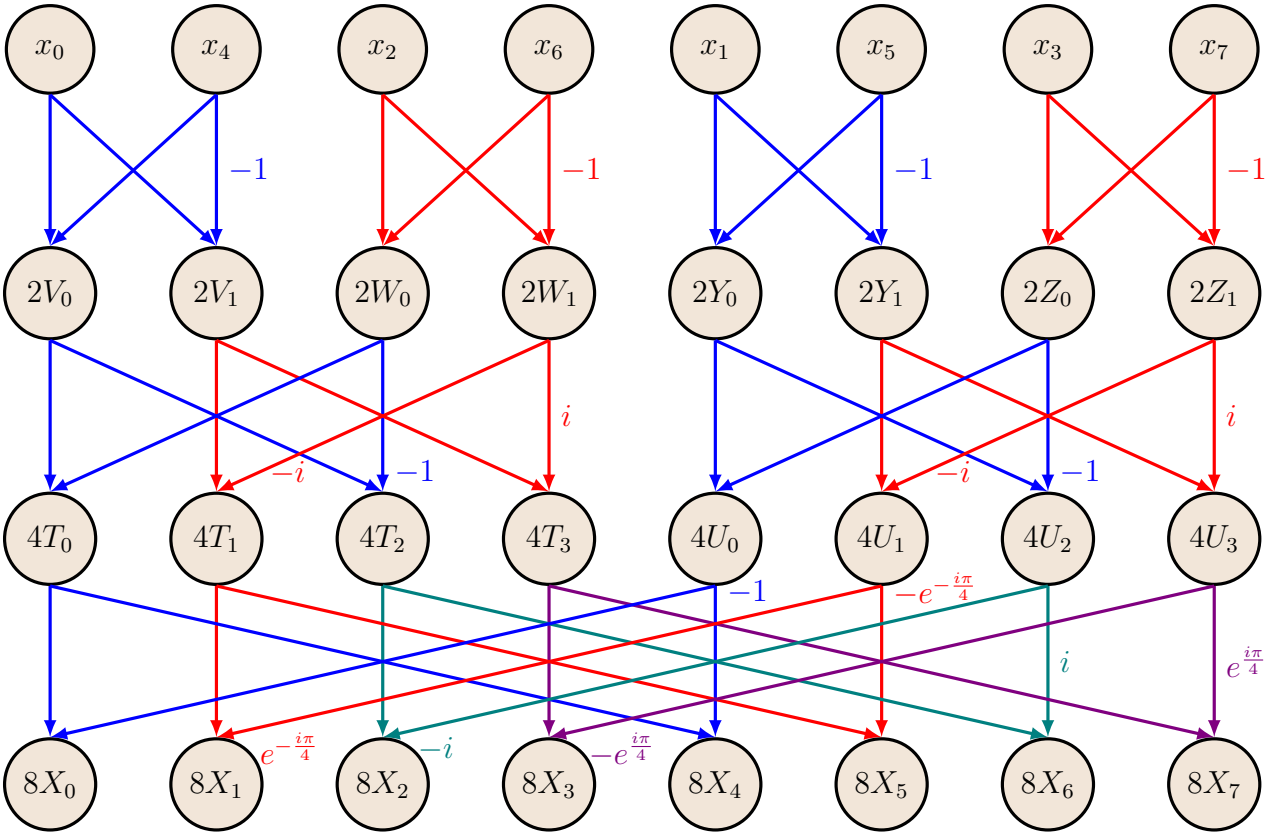


Figura 2.3: Diagrama de flujo de "mariposas" del algoritmo de la *FFT*.

La sucesión de pasos que hemos desarrollado a lo largo del algoritmo de la *FFT* se suele representar de forma esquematizada en unos diagramas de flujo denominados de "mariposas". Estos organigramas tienen gran relevancia a la hora de hacer el seguimiento del algoritmo por etapas. Cada nodo representa un término de la sucesión, y cada flecha e índice asociado señalan la manera en la que se van combinando. En la Figura 2.3, podemos ver las "mariposas" correspondientes al ejemplo desarrollado.

Para una sucesión inicial de 8 términos como la del ejemplo anterior hemos necesitado 24 operaciones (3 etapas y 8 operaciones en cada una de ellas) para aplicar el algoritmo de la *FFT*. En la Tabla 2.1 se recoge una estadística comparativa del coste computacional entre aplicar directamente la *DFT* o utilizar el algoritmo de la *FFT*. Podemos notar la evidente reducción de operaciones que son necesarias con este último método a medida que crece N.

Tal y como hemos mencionado al comienzo de esta sección, el cálculo de la *DFT* de una sucesión de N términos tiene un coste computacional N^2. Por otro lado, el coste de la *FFT* se obtiene de multiplicar N por el número de etapas del algoritmo. Cuando N es una potencia de 2 este número es $\log_2 N$.

Longitud de la sucesión inicial (N)	***DFT* (N^2)**	***FFT* ($N \log_2 N$)**
2	4	2
4	16	8
8	64	24
16	256	64
32	1 024	160
64	4 096	384
128	16 384	896
256	65 536	2 048
512	262 144	4 608
1 024	1 048 576	10 240
2 048	4 194 304	22 528
4 096	16 777 216	49 152

Tabla 2.1: Costes de computación de la *DFT* con o sin el algoritmo de la *FFT* para sucesiones con 2^n términos, $1 \leq n \leq 12$.

Es decir, el algoritmo de la *FFT* permite reducir el coste computacional de N^2 a $N \log_2 N$. Además de la obvia reducción de tiempo de trabajo, hay que destacar que se ejecutan menos operaciones. Esto tiene como consecuencia que los cálculos sean más precisos, ya que hay menos probabilidad de que se produzcan errores de redondeo [13].

Concluyamos esta sección deteniéndonos un poco más en el algoritmo de la *FFT*. La ecuación (2.12) admite una reescritura en forma matricial

$$X = [F_N] \cdot x, \tag{2.15}$$

donde $X = [X_0, X_1, X_2, \ldots, X_{N-1}]^T$, $x = [x_0, x_1, x_2, \ldots, x_{N-1}]^T$ y

$$[F_N] = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{pmatrix}, \qquad W_N = e^{-\frac{2\pi i}{N}}.$$

Notar que la entrada $[F_N]_{ij}$ es W_N^{ij}. El algoritmo de la *FFT* factoriza la matriz $[F_N]$ en un producto de matrices con muchas entradas nulas. Esto es lo que provoca una reducción del coste computacional en el producto $[F_N] \cdot x$. Veamos esta descomposición para el ejemplo de una sucesión de 8 elementos. La factorización de $[F_8]$ es

$$[F_8] = \left(\begin{array}{c|c} [I_4] & [D_4] \\ \hline [I_4] & [-D_4] \end{array}\right) \cdot \left(\begin{array}{c|c} [F_4] & 0 \\ \hline 0 & [F_4] \end{array}\right) \cdot [P_8]\,, \tag{2.16}$$

donde $[I_N]$ es la matriz identidad de tamaño N, $[D_N]$ se define por

$$[D_N] = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & W_{2N} & 0 & \cdots & 0 \\ 0 & 0 & W_{2N}^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & W_{2N}^{N/2-1} \end{pmatrix}$$

y $[P_8]$ es la matriz de permutación

$$[P_8] = \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

En la descomposición de $[F_8]$ aparece dos veces la matriz $[F_4]$, por lo que podemos iterar el proceso anterior y factorizar estas últimas con

$$[F_4] = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -i \\ \hline 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & i \end{array}\right) \cdot \left(\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right) \cdot \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right),$$

resultando X en un producto de matrices dispersas (del inglés *sparse*) o con muchas entradas nulas.

2.4. Transformada de Fourier de coseno

El uso de las series de Fourier está ligado a las funciones periódicas. En el caso de que la función de partida no lo sea, es posible descomponerla en una especie de análogo de (2.1). Suponiendo que la función f es L-periódica, podemos escribir

$$c_n = \frac{1}{L}\int_{-\frac{L}{2}}^{\frac{L}{2}} f(t)e^{-i\left(\frac{2\pi nt}{L}\right)}\,dt, \tag{2.17}$$

donde c_n estaba definido en (2.9).

La expresión (2.8) puede reescribirse como

$$\sum_{n\in\mathbb{Z}} c_n e^{i\left(\frac{2\pi nt}{L}\right)}, \tag{2.18}$$

si asignamos $c_{-n} = a_n + ib_n$, $n \geq 1$. Si la función no es periódica, de manera heurística podemos pensar que su periodo es infinito. Haciendo tender L a infinito en (2.17) y (2.18) obtenemos de forma natural los conceptos de transformada de Fourier y de transformada de Fourier inversa en $\mathbb{R}$.

Definimos la **transformada de Fourier** de una función $f(t)$ integrable Lebesgue en $\mathbb{R}$ como la función $\hat{f}$ definida como

$$\hat{f}(\xi) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(t)e^{-i\xi t}\,dt, \qquad \xi \in \mathbb{R}.$$

La función original f se recupera (en general, esto no es cierto, sino que hay que asumir ciertas condiciones sobre f) a partir de $\hat{f}$ mediante la expresión

$$f(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \hat{f}(\xi)e^{i\xi t}\,d\xi,$$

denominada **transformada de Fourier inversa** (teorema de inversión de Fourier).

Un caso particular de especial interés para el desarrollo de este trabajo es la transformada de Fourier de coseno. Si nos quedamos con la parte de $f(t)$ para $t \geq 0$, podemos construir una nueva función par $g(t)$ (extensión par de f) definida por

$$g(t) = \begin{cases} f(t), & \text{si} \quad t \geq 0, \\ \\ f(-t), & \text{si} \quad t < 0. \end{cases} \tag{2.19}$$

Su transformada de Fourier es

$$\begin{aligned} \hat{g}(\xi) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} g(t)e^{-i\xi t}\,dt \\ &= \frac{1}{\sqrt{2\pi}}\left(\int_0^{\infty} f(t)e^{-i\xi t}dt + \int_{-\infty}^{0} f(-t)e^{-i\xi t}\,dt\right) \end{aligned}$$

$$= \frac{1}{\sqrt{2\pi}} \int_0^\infty f(t) \left(e^{-i\xi t} + e^{i\xi t} \right) \, dt$$
$$= \sqrt{\frac{2}{\pi}} \int_0^\infty f(t) \cos(\xi t) \, d\xi.$$

De esta manera, definimos la **transformada de Fourier de coseno** de una función $f(t)$ apropiada como

$$\hat{f}(\xi) = \sqrt{\frac{2}{\pi}} \int_0^\infty f(t) \cos(\xi t) \, d\xi. \tag{2.20}$$

Con ciertas restricciones podemos recuperar f mediante la **transformada de Fourier de coseno inversa**

$$f(t) = \sqrt{\frac{2}{\pi}} \int_0^\infty \hat{f}(\xi) \cos(\xi t) \, d\xi. \tag{2.21}$$

De cara a la compresión del *JPEG*, nos interesarán especialmente los análogos discretos de las fórmulas (2.20) y (2.21) [5]. Como veremos en la siguiente sección, existen varias discretizaciones posibles, que dan lugar a diferentes transformadas discretas de coseno.

2.5. Transformada discreta de coseno

La *DFT* permite descomponer una señal digital en sus componentes de frecuencia y el algoritmo de la *FFT* posibilita que este proceso se pueda llevar a cabo en un tiempo razonable. Ahora bien, uno de los inconvenientes es que las componentes X_n de la sucesión *DFT* (ver la fórmula (2.10)) son, en general, números complejos. En esta sección introduciremos una familia de transformadas "reales". Una de ellas está en la base de la compresión del *JPEG*.

De forma similar a como se discretizó la expresión (2.9) para obtener el concepto de *DFT*, también podemos discretizar (2.20). Existen varias maneras de hacerlo, lo que da lugar a diferentes versiones de la **transformada discreta de coseno** o *DCT* (*Discrete Cosine Transform*) [15]. Vienen definidas, respectivamente, a partir de las siguientes matrices de transformación:

1. $[B_N]_{nr} = \sqrt{\frac{2}{N}} \left(Q_n Q_r \cos\left(\frac{rn\pi}{N} \right) \right), \qquad n, r \in \mathbb{Z}_{N+1}.$

2. $[C_N]_{nr} = \sqrt{\frac{2}{N}} \left(Q_n \cos\left(\frac{(2r+1)n\pi}{2N} \right) \right), \qquad n, r \in \mathbb{Z}_N.$

3. $[D_N]_{nr} = \sqrt{\frac{2}{N}} \left(Q_r \cos\left(\frac{(2n+1)r\pi}{2N} \right) \right), \qquad n, r \in \mathbb{Z}_N.$

4. $[E_N]_{nr} = \sqrt{\frac{2}{N}} \left(\cos\left(\frac{(2r+1)(2n+1)\pi}{4N} \right) \right), \qquad n, r \in \mathbb{Z}_N.$

donde Q_k es un coeficiente tal que

$$Q_k = \begin{cases} \dfrac{1}{\sqrt{2}}, & \text{si} \quad k \in \{0, N\}\,, \\ \\ 1, & \text{si} \quad k \notin \{0, N\}\,. \end{cases} \tag{2.22}$$

Así las cuatro diferentes DCT son [14]

- *DCT-I*:

$$X_n^{C(1)} = \sqrt{\frac{2}{N}} \sum_{r=0}^{N} Q_n Q_r x_r \cos\left(\frac{rn\pi}{N}\right), \qquad n \in \mathbb{Z}_{N+1}.$$

- *DCT-II*:

$$X_n^{C(2)} = \sqrt{\frac{2}{N}} \sum_{r=0}^{N-1} Q_n x_r \cos\left(\frac{(2r+1)n\pi}{2N}\right), \qquad n \in \mathbb{Z}_N. \tag{2.23}$$

- *DCT-III*:

$$X_n^{C(3)} = \sqrt{\frac{2}{N}} \sum_{r=0}^{N-1} Q_r x_r \cos\left(\frac{(2n+1)r\pi}{2N}\right), \qquad n \in \mathbb{Z}_N.$$

- *DCT-IV*:

$$X_n^{C(4)} = \sqrt{\frac{2}{N}} \sum_{r=0}^{N-1} x_r \cos\left(\frac{(2r+1)(2n+1)\pi}{4N}\right), \qquad n \in \mathbb{Z}_N.$$

La transformada discreta de coseno que utiliza la compresión del *JPEG* es la *DCT-II* y fue introducida en [2, 1974] por N. Ahmed, T. Natarajan y K. R. Rao.

Para hallar la aplicación inversa de esta transformación de $\mathbb{R}^N$ en $\mathbb{R}^N$, demostraremos que su matriz de transformación $[C_N]$ es unitaria. De hecho, esto es equivalente a demostrar que $[C_N]$ es ortogonal, ya que es una matriz real. Con este propósito, denotemos por g_r el r-ésimo vector columna de $[C_N]$; es decir, $g_r = \left[g_{0r}, g_{1r}, \ldots, g_{(N-1)r}\right]^T$, donde

$$g_{nr} = \sqrt{\frac{2}{N}} \left(Q_n \cos\left(\frac{(2r+1)n\pi}{2N}\right)\right), \qquad n \in \mathbb{Z}_N. \tag{2.24}$$

Recordemos primero unos resultados sencillos que se van a utilizar.

1. Suma de progresiones geométricas finitas:

$$S = \sum_{n=-N}^{N} w^n = \begin{cases} 2N+1, & \text{si} \quad w = 1, \\ \\ \dfrac{w^{-N} - w^{N+1}}{1-w}, & \text{si} \quad w \neq 1. \end{cases} \tag{2.25}$$

2. En particular, tomando $w = e^{ix}$ y simplificando (2.25) mediante $\sin(x) = \frac{e^{ix}-e^{ix}}{2i}$, obtenemos

$$\sum_{n=-N}^{N} e^{inx} = \begin{cases} 2N+1, & \text{si} \quad e^{ix} = 1, \\ \\ \dfrac{\sin\left((N+\frac{1}{2})x\right)}{\sin\left(\frac{x}{2}\right)}, & \text{si} \quad e^{ix} \neq 1. \end{cases} \tag{2.26}$$

3. Poniendo $\cos(x) = \frac{e^{ix}+e^{-ix}}{2}$ en la ecuación (2.26) obtenemos

$$\sum_{n=1}^{N} \cos(nx) = \frac{1}{2}\sum_{n=-N}^{N} e^{inx} - \frac{1}{2} = \begin{cases} N, & \text{si} \quad e^{ix} = 1, \\ \\ \dfrac{\sin\left((N+\frac{1}{2})x\right)}{2\sin\left(\frac{x}{2}\right)} - \dfrac{1}{2}, & \text{si} \quad e^{ix} \neq 1. \end{cases} \tag{2.27}$$

Notar que la condición $e^{ix} = 1$ es equivalente a $x \equiv 0 \pmod{2\pi}$.

Volviendo a la demostración, el producto interno de dos vectores reales g_p y g_q en $\mathbb{R}^N$ viene dado por la fórmula

$$\langle g_p, g_q \rangle = \sum_{n=0}^{N-1} g_{np} \cdot g_{nq}. \tag{2.28}$$

Queremos probar que

$$\langle g_p, g_q \rangle = \begin{cases} 1, & \text{si} \quad p = q, \\ \\ 0, & \text{si} \quad p \neq q. \end{cases} \tag{2.29}$$

Utilizando (2.28) y (2.24)

$$\begin{aligned} \langle g_p, g_q \rangle =& \frac{2}{N}\sum_{n=0}^{N-1} {Q_n}^2 \cos\left(\frac{(2p+1)n\pi}{2N}\right)\cos\left(\frac{(2q+1)n\pi}{2N}\right) \\ =& \frac{2}{N}\cdot\left(\frac{1}{\sqrt{2}}\right)^2 + \frac{1}{N}\sum_{n=1}^{N-1} 2\cos\left(\frac{(2p+1)n\pi}{2N}\right)\cos\left(\frac{(2q+1)n\pi}{2N}\right) \\ =& \frac{1}{N}\left(1 + \sum_{n=1}^{N-1}\cos\left((p+q+1)\frac{n\pi}{N}\right) + \sum_{n=1}^{N-1}\cos\left((p-q)\frac{n\pi}{N}\right)\right). \end{aligned}$$

En este punto, podemos distinguir tres casos. Utilizaremos la ecuación (2.27) presentada anteriormente para simplificar los resultados en cada caso.

- Si $p = q$, entonces

$$\langle g_p, g_p \rangle = \frac{1}{N}\left(1 + \sum_{n=1}^{N-1}\cos\left((2p+1)\frac{n\pi}{N}\right) + (N-1)\right).$$

Y como $2p+1$ es un número impar,

$$\langle g_p, g_q \rangle = \frac{1}{N}(1 + 0 + N - 1) = 1.$$

- Si $p \neq q$ y $p+q$ es par, entonces $p-q$ es par y $p+q+1$ es impar. Luego,

$$\begin{aligned}\langle g_p, g_q\rangle =& \frac{1}{N}\left(1+\sum_{n=1}^{N-1}\cos\left((p+q+1)\frac{n\pi}{N}\right)+\sum_{n=1}^{N-1}\cos\left((p-q)\frac{n\pi}{N}\right)\right)\\ =& \frac{1}{N}\left(1+0+(-1)\right)=0.\end{aligned}$$

- Si $p \neq q$ y $p+q$ es impar, entonces $p-q$ es impar y $p+q+1$ es par. Entonces,

$$\begin{aligned}\langle g_p, g_q\rangle =& \frac{1}{N}\left(1+\sum_{n=1}^{N-1}\cos\left((p+q+1)\frac{n\pi}{N}\right)+\sum_{n=1}^{N-1}\cos\left((p-q)\frac{n\pi}{N}\right)\right)\\ =& \frac{1}{N}\left(1+(-1)+0\right)=0.\end{aligned}$$

De esta manera, hemos probado (2.29).

Así, la matriz de transformación $[C_N]$ de la *DCT-II* es ortogonal y se cumple que

$$[C_N]^{-1} = [C_N]^T.$$

Resulta que la traspuesta de esta matriz tiene por entradas

$$[C_N]^T_{nr} = \left[\sqrt{\frac{2}{N}}\left(Q_n\cos\left(\frac{(2r+1)n\pi}{2N}\right)\right)\right]^T = \sqrt{\frac{2}{N}}\left(Q_r\cos\left(\frac{(2n+1)r\pi}{2N}\right)\right) = [D_N]_{nr},$$

por lo que

$$[C_N]^{-1} = [C_N]^T = [D_N],$$

donde $[D_N]$, tal y como hemos visto anteriormente, es la matriz de transformación de la *DCT-III*.

Si repetimos análogamente estos pasos con el resto de transformadas llegamos a las conclusiones siguientes

1. $[B_N]^{-1} = [B_N]^T = [B_N]$.
2. $[C_N]^{-1} = [C_N]^T = [D_N]$.
3. $[D_N]^{-1} = [D_N]^T = [C_N]$.
4. $[E_N]^{-1} = [E_N]^T = [E_N]$.

En resumen, obtenemos la siguientes expresiones para las **transformadas discretas de coseno inversas** de *DCT-I*, *DCT-II*, *DCT-III* y *DCT-IV*, respectivamente [14]:

- *IDCT-I*:

$$x_r = \sqrt{\frac{2}{N}}\sum_{n=0}^{N} Q_r Q_n X_n \cos\left(\frac{rn\pi}{N}\right), \qquad r \in \mathbb{Z}_{N+1}.$$

- *IDCT-II*:

$$x_r = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} Q_n X_n \cos\left(\frac{(2r+1)n\pi}{2N}\right), \qquad r \in \mathbb{Z}_N. \tag{2.30}$$

- *IDCT-III*:

$$x_r = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} Q_r X_n \cos\left(\frac{(2n+1)r\pi}{2N}\right), \qquad r \in \mathbb{Z}_N.$$

- *IDCT-IV*:

$$x_r = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} X_n \cos\left(\frac{(2r+1)(2n+1)\pi}{4N}\right), \qquad r \in \mathbb{Z}_N.$$

Como ya hemos advertido, en este trabajo nos interesa especialmente la *IDCT-II*.

2.6. Algoritmo de la transformada rápida de coseno

En la sección 2.3 vimos como se podía utilizar el algoritmo de la transformada rápida de Fourier para calcular de forma más eficiente la *DFT* de una señal. Resulta natural preguntarse si existen algoritmos de este tipo para las transformadas discretas de coseno de la sección anterior. La respuesta es afirmativa y en esta sección veremos dos para la *DCT-II*. En el primero se calculará la *DCT-II* de una señal a través de la *FFT* [2]. La idea para hacer esto proviene de la relación (2.19) entre la transformada de Fourier de una función par y la transformada de Fourier de coseno. Veámoslo a continuación.

Supongamos que queremos calcular la *DCT-II* de una sucesión $\{x_r\}_{r\in\mathbb{Z}_N}$. Definamos para ello otra sucesión $\{y_r\}_{r\in\mathbb{Z}_{2N}}$ como su extensión simétrica en torno al punto $N - \frac{1}{2}$, es decir,

$$y_r = \begin{cases} x_r, & \text{si} \quad r \in \mathbb{Z}_N, \\ \\ x_{2N-r-1}, & \text{si} \quad r \in \mathbb{Z}_{2N} \setminus \mathbb{Z}_N. \end{cases}$$

La *DFT* de $\{y_r\}_{r\in\mathbb{Z}_{2N}}$ viene dada por

$$\begin{aligned} Y_n &= \frac{1}{2N} \sum_{r=0}^{2N-1} y_r e^{-i\left(\frac{2\pi nr}{2N}\right)}, \\ &= \frac{1}{2N} \sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{2N}\right)} + \frac{1}{2N} \sum_{r=N}^{2N-1} x_{2N-r-1} e^{-i\left(\frac{2\pi nr}{2N}\right)}, \\ &= \frac{1}{2N} \sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi nr}{2N}\right)} + \frac{1}{2N} \sum_{r=0}^{N-1} x_r e^{-i\left(\frac{2\pi n(2N-r-1)}{2N}\right)}. \end{aligned}$$

Como $e^{-i\left(\frac{2\pi n(2N)}{2N}\right)} = 1$, llegamos a que

$$Y_n = \frac{1}{2N} \sum_{r=0}^{N-1} x_r \left(e^{-i\left(\frac{2\pi nr}{2N}\right)} + e^{i\left(\frac{2\pi n(r+1)}{2N}\right)}\right), \qquad n \in \mathbb{Z}_{2N}.$$

Ahora, multiplicamos ambos lados de la igualdad por $\sqrt{2N}Q_n e^{-i\left(\frac{\pi n}{2N}\right)}$, donde Q_n está definido por (2.22), y obtenemos

$$\sqrt{2N}Q_n e^{-i\left(\frac{\pi n}{2N}\right)}Y_n = \sqrt{\frac{2}{N}}\sum_{r=0}^{N-1} Q_n x_r \cos\left(\frac{(2r+1)n\pi}{2N}\right), \qquad n \in \mathbb{Z}_N.$$

Si comparamos esta expresión con la ecuación (2.23), notamos que la parte derecha de la igualdad es la fórmula de *DCT-II*. A través de esta relación entre *DFT* y *DCT-II*, podemos aplicar el algoritmo de la *FFT*, que hemos visto en (2.14), para calcular la *DCT-II* de una señal $\{x_r\}_{r\in\mathbb{Z}_{2N}}$.

No obstante, este algoritmo no es lo suficientemente eficiente para el cálculo de la *DCT-II* en muchas de las aplicaciones. El primer algoritmo que no utilizó la *FFT* para calcular la *DCT-II* tiene sus bases en el artículo [6, 1977] de W. H. Chen, C. H. Smith y S. C. Fralick. Este algoritmo, que es seis veces más rápido que el que hace uso de la *FFT*, se basa en la siguiente descomposición matricial de la *DCT-II* (similar a (2.15)):

$$X = \sqrt{\frac{2}{N}}A_N \cdot x, \tag{2.31}$$

donde $X = (X_0, X_1, X_2, \ldots, X_{N-1})^T$, $x = (x_0, x_1, x_2, \ldots, x_{N-1})^T$ y

$$[A_N]_{nr} = \sqrt{\frac{N}{2}}\,[C_N]_{nr} = Q_n \cos\left(\frac{(2r+1)n\pi}{2N}\right), \qquad n, r \in \mathbb{Z}_N.$$

Presentemos el caso particular $N = 8$. Dada una sucesión $\{x_r\}_{r=0}^7$, calculamos su *DCT-II* $\{X_n\}_{n=0}^7$ mediante

$$X_n^{C(2)} = \frac{1}{2}\sum_{r=0}^{7} Q_n x_r \cos\left(\frac{(2r+1)n\pi}{16}\right), \qquad n \in \mathbb{Z}_8,$$

con Q_n definido por (2.22).

Análogamente a lo explicado en la sección 2.3, el objetivo de este algoritmo rápido es factorizar la matriz $[A_N]$ de la ecuación (2.31) como producto de matrices dispersas, es decir, matrices donde la mayoría de sus entradas son nulas. Esto se consigue de una manera recurrente. Si $N = 8$,

$$[A_8] = [P_8] \cdot \left(\begin{array}{c|c} [A_4] & 0 \\ \hline 0 & \left[\overline{R_4}\right] \end{array}\right) \cdot [B_8]\,, \tag{2.32}$$

donde $[P_8]$ es la matriz de permutación

$$[P_8] = \left(\begin{array}{cccc:cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right),$$

$\left[\overline{R_4}\right] = \left[\overline{I_4}\right] \cdot [R_4] \cdot \left[\overline{I_4}\right]$, donde, en general,

$$\left[\overline{I_N}\right] = \begin{pmatrix} 0 & \cdots & 0 & 0 & 1 \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & 1 & 0 & 0 \\ \vdots & \iddots & \vdots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 & 0 \end{pmatrix} \qquad \text{y} \qquad [R_N]_{rn} = \cos\left(\frac{(2r+1)(2n+1)\pi}{4N}\right),$$

y $[B_8]$ viene dada por

$$[B_8] = \left(\begin{array}{c:c} [I_4] & \left[\overline{I_4}\right] \\ \hdashline \left[\overline{I_4}\right] & [-I_4] \end{array}\right) = \left(\begin{array}{cccc:cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{array}\right).$$

La factorización de $[R_4]$ es

$$[R_4] = [Q_4] \cdot [V_4(2)] \cdot [T_4] \cdot [V_4(2)]\,,$$

con

$$[Q_4] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad [V_4(2)] = \begin{pmatrix} \cos\frac{\pi}{16} & 0 & 0 & \sin\frac{\pi}{16} \\ 0 & \sin\frac{3\pi}{16} & \cos\frac{3\pi}{16} & 0 \\ 0 & \cos\frac{3\pi}{16} & -\sin\frac{3\pi}{16} & 0 \\ \sin\frac{\pi}{16} & 0 & 0 & -\cos\frac{\pi}{16} \end{pmatrix},$$

$$[T_4] = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \qquad [V_4(1)] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\frac{\pi}{4} & \cos\frac{\pi}{4} & 0 \\ 0 & \cos\frac{\pi}{4} & -\cos\frac{\pi}{4} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

En el segundo factor de (2.32) ha aparecido la matriz $[A_4]$. Así, de una manera recurrente,

$$[A_4] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \left(\begin{array}{c|c} [A_2] & 0 \\ \hline 0 & \left[\overline{R_2}\right] \end{array}\right) \cdot \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix},$$

y con esto se logra el objetivo de reducir la *DCT-II* a un producto de matrices dispersas.

Capítulo 3

Compresión del *JPEG*

El proceso de compresión y codificación de imágenes que diseñó el grupo *JPEG* se recoge en el estándar [16]. Allí se distinguen, como hemos dicho, dos clases de compresión (con pérdida y sin pérdida) y además cuatro modelos de codificación (*JPEG sequential DCT-based coding models*, *models for progressive DCT-based coding*, *coding models for lossless coding*, y *models for hierarchical coding*). En este capítulo comenzaremos con unas breves pinceladas históricas del *JPEG* [11], para después pasar a describir el denominado *baseline sequential process*, un proceso de compresión con pérdida basado en el primero de los modelos de codificación mencionados. Este proceso es probablemente el más extendido a la hora de comprimir imágenes.

3.1. Breve historia del *JPEG*

En la década de 1980, la disciplina de las telecomunicaciones giró en torno al lanzamiento a nivel mundial de un innovador servicio conocido como *videotex*, que permitía a los usuarios obtener acceso a información y contenido multimedia mediante el uso de terminales de computadora y televisión. Poco tiempo después surgió la tecnología conocida como red digital de servicios integrados (*ISDN*) que dio lugar a la aparición de las primeras máquinas con pantallas compatibles con gráficos y color. Sin embargo, a pesar de representar un gran avance, estos dispositivos carecían de la capacidad de mostrar imágenes fotográficas en ese momento.

Con la llegada de la *ISDN*, los centros de investigación de las principales empresas de telecomunicaciones se centraron en encontrar maneras de mejorar la capacidad de visualización a través de técnicas de codificación de imágenes fotográficas. Sin embargo, el problema era que las imágenes fotográficas contienen mucha información, lo que provocó la necesidad de buscar una forma de reducir su tamaño para acortar el tiempo de transmisión de las mismas.

La formación del proyecto *European Photovideotex Image Compression Algorithms* (*PICA*) en 1985, a partir de un consorcio de socios experimentados en telecomunicaciones,

computación y radiodifusión, resultó ser un gran impulso para el desarrollo internacional de las técnicas de codificación de imágenes. Durante cuatro años, el proyecto *PICA* realizó contribuciones clave en el desarrollo de requisitos técnicos y en la evaluación de técnicas de codificación que en el futuro servirían de base para la formación del estándar *JPEG*.

Figura 3.1: Comité del *JPEG*.

Hasta entonces los organismos internacionales encargados de regular y normalizar el *videotex* eran el Comité Consultivo Internacional Telegráfico y Telefónico (*CCITT*), la Conferencia Europea de Administraciones de Correos y Telecomunicaciones (*CEPT*) y la Organización Internacional de Normalización (*ISO*).

Tras darse cuenta de la relevancia que iba a tener la codificación de imágenes en los futuros servicios de comunicación multimedia, los líderes de las organizaciones *CCITT* e *ISO* (*Working Group 8* o *WG8*) unieron fuerzas a mediados de 1986 para trabajar de forma conjunta, dando origen al *Joint Photographic Experts Group* (*JPEG*). Este comité tenía el objetivo de, a partir de los requisitos de servicio establecidos por el *CCITT* y la experiencia de codificación de la *ISO*, encontrar y estandarizar una técnica de alto rendimiento de compresión de imágenes fotográficas.

Con este fin se definieron las características esenciales que debía de tener el procedimiento que se iba a desarrollar. La primera de ellas era la capacidad de ajustar el factor de compresión con respecto a la calidad final obtenida. Es decir, la posibilidad de regular según las necesidades de la aplicación cuánto reducir los datos en función de cuánto se deba parecer el resultado a la imagen original. La segunda era la creación progresiva de imágenes, justificada por la restricción de velocidad en los canales de transmisión. Este concepto consistía en entregar rápidamente una imagen para visualización instantánea que tuviera una resolución mucho menor, para que, posteriormente, en varias etapas se pudiera mejorar hasta lograr la máxima calidad. La tercera y última, también con el fin de mejorar la velocidad de transferencia de imágenes, era la propiedad de construirse secuencialmente línea por línea de arriba a abajo.

Para encontrar la técnica de codificación que más se adecuase a sus intereses, el grupo de trabajo *JPEG* convocó un concurso público. Los candidatos debían proporcionar una descripción técnica completa de su método y, además, tenían que realizar diferentes compresiones y descompresiones con dicho algoritmo con diferentes factores de compresión sobre un conjunto establecido de imágenes de prueba. De las doce propuestas que se registraron oficialmente en marzo de 1987, llegaron tres a la selección final en junio de 1987. Tras varios meses de pruebas, en la reunión de enero de 1988 en Copenhague se proclamó ganadora por unanimidad la técnica adaptativa de transformada de coseno (*Adaptative Discrete Cosine Transform* o *ADCT*) planteada por el *PICA*.

Tras esta decisión, el comité *JPEG* decidió desarrollar un estándar basado en esta técnica *ADCT*, en la que se aplicaban diferentes claves científicas que veremos desarrolladas en mayor profundidad en la siguiente sección. Entre ellas se debe destacar alguna, como la transformada discreta de coseno, la codificación de bloques o la cuantización psicovisual. Por último, se debe señalar que es notable la capacidad de evolucionar que ha tenido esta técnica de compresión a lo largo de los años, siendo el algoritmo *JPEG* tiempo después la base de otras extensiones como el *JPEG 2000* o el *JPEG XT*. Además, *JPEG* sirvió como modelo para otro tipo de estándares de compresión como, por ejemplo, *Motion Picture Experts Group* (*MPEG*), tan extendido en la manipulación de archivos de vídeo.

3.2. Descomposición en bloques y tratamiento del color

Como hemos mencionado anteriormente, una imagen (en blanco y negro) puede entenderse como una señal bidimensional digital. El ordenador la interpreta como un rectángulo formado por píxeles, que es su menor unidad homogénea. El método de compresión del *JPEG* comienza dividiendo la imagen a comprimir en bloques de tamaño 8×8 píxeles llamados *MCU* (*Minimum Coded Units*). Cuando las dimensiones no son múltiplos de 8 se aplican técnicas de relleno para ajustar la imagen al tamaño adecuado. Estos bloques van a ser tratados de forma independiente, por lo que de aquí en adelante nos centraremos en el proceso de compresión que se realiza en uno de ellos, ya que este método se repetirá de forma análoga con el resto de bloques. En la Figura 3.2 podemos ver la imagen de la Catedral de Burgos dividida en bloques de 8×8 píxeles y en la Figura 3.3 se muestra en detalle uno de esos *MCU*, donde cada cuadrado de la rejilla es un píxel de la imagen.

Después de haber separado la imagen en *MCU*, se descompone cada uno de ellos en sus valores *RGB*, es decir, cada *MCU* proporciona tres bloques, uno para el color rojo, otro para el verde y otro para el azul. El color de un píxel se puede determinar a través de varios estándares. De momento solo hemos tratado el modelo de color *RGB* (*Red*, *Green* y *Blue*), que probablemente es el más popular. Cada color se construye como una mezcla por adición de esos tres colores de luz primarios. En este sistema, como ya dijimos en la introducción, el

Figura 3.2: Imagen de la Catedral de Burgos dividida en *MCU*. Hemos seleccionado uno de ellos como muestra.

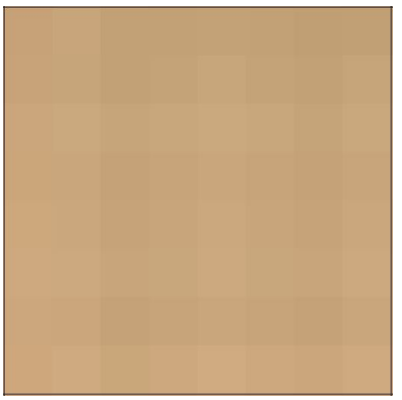


Figura 3.3: *MCU* seleccionado de la imagen de la Catedral de Burgos.

color de cada píxel se determina con un vector de tres parámetros que toman un valor entero de 0 a 255 y que representan las cantidades de rojo, verde y azul que contiene dicho píxel. Como cada parámetro puede tomar 256 valores y $256 = 2^8$, para representar dicho parámetro se requieren ocho bits, o lo que es lo mismo, un byte. Por tanto, para almacenar el color de un píxel se necesitan tres bytes, y como un *MCU* tiene 64 píxeles, utilizaremos $64 \times 3 = 192$ bytes para guardar un bloque en memoria.

En la Figura 3.4 podemos apreciar qué representa cada parámetro del espacio de color *RGB* en nuestra imagen de la Catedral de Burgos. Si combinásemos los tres canales (*Red*, *Green* y *Blue*) obtendríamos la imagen original. Sin embargo, cuando un ordenador procesa una imagen en un espacio de color concreto, no es así como lo almacena. En realidad, sim-

(a) *Red* (b) *Green* (c) *Blue*

Figura 3.4: Espacio de color *RGB* de la imagen de la Catedral de Burgos.

plemente entiende cada parámetro del espacio de color como una imagen en escala de grises. En esta escala monocromática únicamente se representa la cantidad de luz o intensidad que tiene cada píxel. Los valores bajos indican píxeles muy oscuros o de baja intensidad, mientras que los valores altos hacen referencia a píxeles claros o con una alta intensidad. En la Figura 3.5 podemos ver la descomposición del ejemplo en este espacio de color.

(a) R (b) G (c) B

Figura 3.5: Descomposición de la imagen de la Catedral de Burgos en el sistema *RGB*.

Por la propia naturaleza de este estándar, el vector tridimensional *RGB* tiene una alta correlación en sus componentes, lo que dificulta la tarea de comprimir la imagen, ya que la pérdida de información en uno de los parámetros afecta de forma relevante a los otros dos parámetros del modelo. Por ello, se realiza una transformación del espacio de color de cada píxel para convertir este vector en uno nuevo, compuesto por otros tres valores. Estos hacen referencia a la luminancia o luma (Y), que describe el brillo de la imagen (las regiones más brillantes son representadas por números más grandes), y las crominancias o cromas de azul (C_b) y de rojo (C_r), que representan los matices rojizos y azulados (ver Figura 3.6).

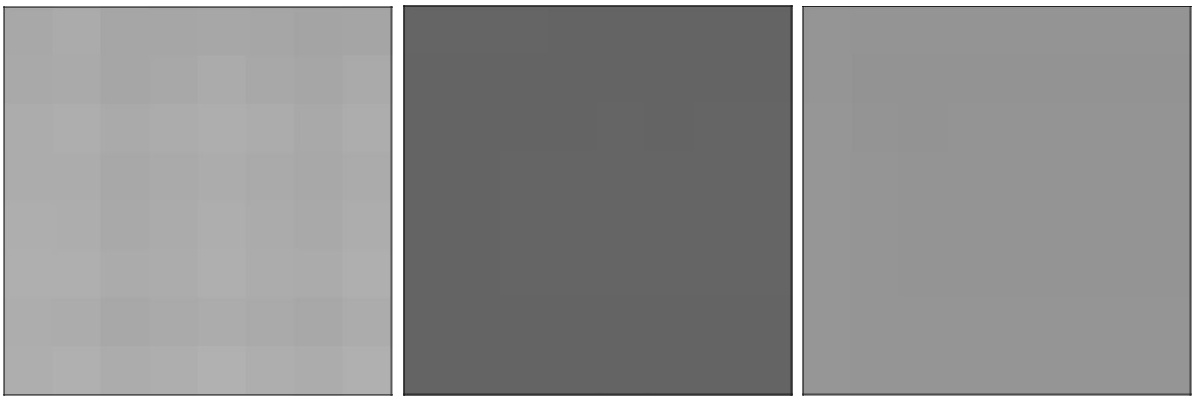
(a) Y (b) C_b (c) C_r

Figura 3.6: Descomposición de la imagen de la Catedral de Burgos en el sistema YC_bC_r.

Para hacer esta transformación de RGB a YC_bC_r se utiliza el siguiente cambio [3] (sumamos 128 a ambas crominancias después del intercambio para obtener valores entre 0 y 255):

$$\begin{pmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{pmatrix} = \begin{pmatrix} 0,29900 & 0,58700 & 0,11400 \\ -0,16874 & -0,33126 & 0,50000 \\ 0,50000 & -0,41869 & -0,08131 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \tag{3.1}$$

Esta transformación no ha reducido el espacio necesario para almacenar un bloque (al ser igualmente tres parámetros seguimos necesitando 192 bytes para guardar un MCU). Sin embargo, este sistema posee dos características psicovisuales que justifica su uso por el $JPEG$ [4]. La primera es que numerosos estudios han demostrado que el ojo humano tiende a ser más sensible a los cambios en la luminancia que en las crominancias [9]. Esto permite realizar mayores cambios en las crominancias sin que nuestra percepción de la imagen se vea afectada. La segunda es que, en general, hay más variación en la luminancia que en las crominancias en un MCU. Esto lo podemos apreciar a modo de ejemplo en las Figuras 3.3 y 3.7 que representan el MCU que habíamos seleccionado anteriormente en la Figura 3.2.



(a) Y (b) C_b (c) C_r

Figura 3.7: Descomposición del MCU seleccionado en las componentes del sistema YC_bC_r.

Atendiendo a estos dos factores, se utiliza una técnica de compresión llamada submuestreo de crominancia (*chroma subsampling*), que consiste en agrupar píxeles vecinos compartiendo información de color entre ellos. El objetivo es reducir la resolución espacial de los canales de croma en comparación con el de luminancia. Existen varios métodos para aplicar esta técnica y no vamos a explicarlos en este trabajo.

La descomposición RGB del MCU de la Figura 3.3 es

$$[R] = \begin{pmatrix} 199 & 200 & 194 & 194 & 195 & 194 & 192 & 193 \\ 200 & 198 & 194 & 196 & 199 & 196 & 194 & 198 \\ 203 & 202 & 198 & 200 & 202 & 200 & 197 & 201 \\ 203 & 201 & 197 & 199 & 201 & 199 & 197 & 200 \\ 205 & 202 & 198 & 200 & 203 & 200 & 198 & 202 \\ 206 & 204 & 200 & 200 & 204 & 200 & 200 & 204 \\ 204 & 203 & 197 & 199 & 201 & 199 & 197 & 201 \\ 206 & 207 & 201 & 205 & 208 & 205 & 203 & 207 \end{pmatrix}, \quad [G] = \begin{pmatrix} 162 & 165 & 161 & 161 & 162 & 161 & 159 & 160 \\ 163 & 165 & 161 & 163 & 166 & 163 & 161 & 165 \\ 166 & 169 & 165 & 167 & 169 & 167 & 164 & 168 \\ 166 & 166 & 162 & 164 & 166 & 164 & 162 & 165 \\ 168 & 167 & 163 & 165 & 168 & 165 & 163 & 167 \\ 169 & 169 & 165 & 167 & 169 & 167 & 165 & 169 \\ 167 & 166 & 162 & 164 & 166 & 164 & 162 & 166 \\ 167 & 170 & 167 & 168 & 171 & 168 & 166 & 170 \end{pmatrix},$$

$$[B] = \begin{pmatrix} 120 & 123 & 118 & 118 & 119 & 118 & 116 & 117 \\ 121 & 122 & 118 & 120 & 123 & 120 & 118 & 122 \\ 124 & 126 & 122 & 124 & 126 & 124 & 121 & 125 \\ 122 & 124 & 120 & 122 & 124 & 122 & 120 & 123 \\ 124 & 125 & 121 & 123 & 126 & 123 & 121 & 125 \\ 127 & 127 & 123 & 124 & 127 & 124 & 123 & 127 \\ 125 & 124 & 120 & 122 & 124 & 122 & 120 & 124 \\ 124 & 128 & 122 & 126 & 129 & 126 & 124 & 128 \end{pmatrix}.$$

Mediante la transformación (3.1) cambiamos el espacio de color al sistema YC_bC_r y obtenemos

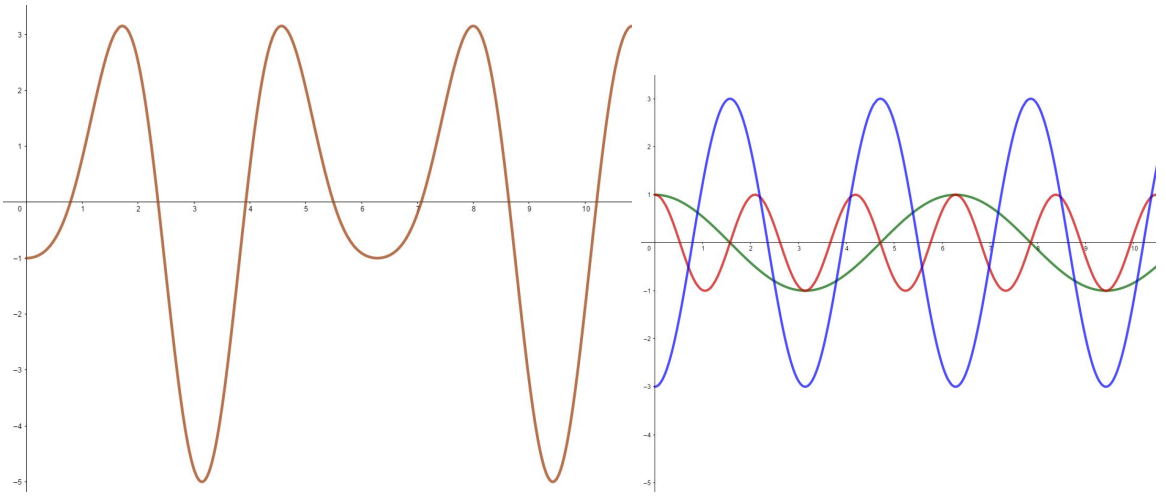
$$[Y] = \begin{pmatrix} 168 & 171 & 166 & 166 & 167 & 166 & 164 & 165 \\ 169 & 170 & 166 & 168 & 171 & 168 & 166 & 170 \\ 172 & 174 & 170 & 172 & 174 & 172 & 169 & 173 \\ 172 & 172 & 168 & 170 & 172 & 170 & 168 & 171 \\ 174 & 173 & 169 & 171 & 174 & 171 & 169 & 173 \\ 175 & 175 & 171 & 172 & 175 & 172 & 171 & 175 \\ 173 & 172 & 168 & 170 & 172 & 170 & 168 & 172 \\ 174 & 176 & 172 & 174 & 177 & 174 & 172 & 176 \end{pmatrix}, \quad [C_b] = \begin{pmatrix} 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 100 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 100 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 100 & 101 & 100 & 101 & 101 & 101 & 101 & 101 \end{pmatrix},$$

$$[C_r] = \begin{pmatrix} 150 & 149 & 148 & 148 & 148 & 148 & 148 & 148 \\ 150 & 148 & 148 & 148 & 148 & 148 & 148 & 148 \\ 150 & 148 & 148 & 148 & 148 & 148 & 148 & 148 \\ 150 & 149 & 149 & 149 & 149 & 149 & 149 & 149 \\ 150 & 149 & 149 & 149 & 149 & 149 & 149 & 149 \\ 150 & 149 & 149 & 148 & 149 & 148 & 149 & 149 \\ 150 & 150 & 149 & 149 & 149 & 149 & 149 & 149 \\ 151 & 150 & 149 & 150 & 150 & 150 & 150 & 150 \end{pmatrix}. \tag{3.2}$$

Se puede comprobar que las tres matrices de (3.2) son congruentes con lo mencionado anteriormente, ya que hay mayor variedad en los valores de la luminancia que en las crominancias.

3.3. Aplicación de la transformada discreta de coseno

Vamos a transformar los valores de color de las tres matrices $[Y]$, $[C_b]$ y $[C_r]$ a un dominio de frecuencias. La serie de Fourier (2.7) es una descomposición de una función periódica como una suma (infinita) de senos y cosenos de diferentes frecuencias (ver Figura 3.8). Los coeficientes de Fourier a_n y b_n son las amplitudes de estas ondas componentes, e indican su grado de contribución a la hora de reconstruir o sintetizar la función. Es decir, miden cuánto se parece la función original a la correspondiente onda sinusoidal.



(a) Ejemplo función periódica (b) Ejemplo descompuesto en ondas

Figura 3.8: Descomposición de Fourier de una función periódica.

Al tratar nuestras señales digitales con la *DCT-II*, es común muestrearlos en la red de 8 puntos equiespaciados $\frac{\pi n}{16}$ $(n \in \mathbb{Z}_8)$ de modo que el papel de la base de ondas sinusoidales de la serie de Fourier lo adquirirán ahora las 8 señales digitales $\cos(rx)$, con $r \in \mathbb{Z}_8$ [1].

Otro detalle a tener en cuenta es que el tratamiento de imágenes se hace analizando señales digitales bidimensionales, por lo que las 8 señales digitales unidimiensionales anteriores dan lugar a $8 \times 8 = 64$ señales digitales bidimiensionales (ver Figura 3.9). Con ellas es posible representar cualquier señal muestreada por 64 valores mediante combinaciones lineales. Los coeficientes necesarios se calculan con el análogo bidimensional de la *DCT-II* (3.9).

De manera general, dada

$$f : \{0, \ldots, M-1\} \times \{0, \ldots, N-1\} \to \{0, 1, \ldots, 255\},$$

se define la transformada discreta de coseno bidimensional (*2D DCT-II*) como la sucesión $\{F_{uv}\}$, $u = 0, \ldots, M-1$; $v = 0, \ldots, N-1$, donde

$$F_{uv} = \frac{2Q_uQ_v}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f_{xy} \cos\left(\frac{(2x+1)u\pi}{2M}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right), \tag{3.3}$$

y Q_n está definido en (2.22).

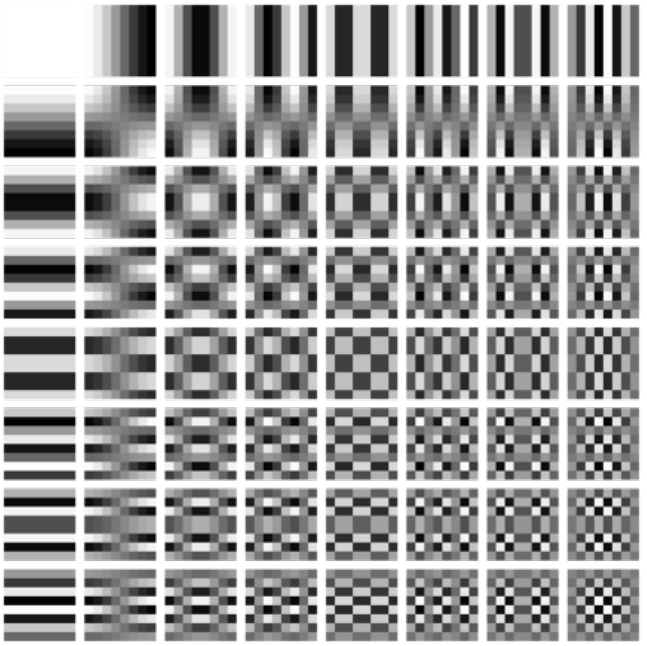


Figura 3.9: Ondas coseno en las que se puede descomponer un *MCU* con la *DCT*.

Esbocemos la aplicación de la *2D DCT-II* en el proceso de compresión del *JPEG*. La Figura 3.9 está constituida por 64 *MCU*, donde el bloque situado en la esquina superior izquierda representa que no hay ninguna variación con los vecinos (onda bidimensional con menor frecuencia). Según nos desplacemos hacia la derecha y descendamos, las frecuencias horizontales y verticales van aumentando de manera gradual, respectivamente. De esta forma, el *MCU* de la esquina inferior derecha representa que en ese bloque los valores han cambiado drásticamente en ambas direcciones con respecto a sus bloques vecinos (onda con mayor frecuencia de todas).

Para aplicar la *2D DCT-II* a cada uno de los bloques *MCU* 8×8, hacemos $M = N = 8$ en (3.3). Si denotamos por f_{xy} el elemento en la posición (x, y) de cada una de las matrices $[Y]$, $[C_b]$ y $[C_r]$, queda

$$F_{uv} = \frac{Q_u^2}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} f_{xy} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right).$$

En nuestro ejemplo particular, al aplicar la *2D DCT-II* a la matriz $[Y]$ de (3.2) obtenemos

$$[F_Y] = \begin{pmatrix} 2735 & 7 & 5 & 6 & 14 & -19 & -5 & -4 \\ -29 & 3 & -2 & 0 & -4 & 0 & -1 & -1 \\ -10 & 1 & -1 & -1 & -3 & -1 & -1 & -1 \\ -11 & 4 & 3 & 1 & -1 & 1 & 0 & 1 \\ -2 & 2 & -1 & 0 & -2 & 0 & -1 & 0 \\ -10 & 4 & 1 & 0 & -2 & 2 & 0 & -1 \\ 15 & 1 & -1 & -2 & -2 & -1 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}. \tag{3.4}$$

Como la *2D DCT-II* es separable, es posible aplicar una *1D DCT-II* a cada una de las filas y luego otra a la columna de las transformadas resultantes por medio de algún algoritmo rápido como el que hemos visto en la sección 2.6.

3.4. Proceso de cuantización

El resultado de haber aplicado la *2D DCT-II* es que cada *MCU* almacenará ahora los coeficientes F_{uv} y, por tanto, la interpretación del bloque es distinta. Antes, cada casilla nos indicaba la magnitud del brillo y los matices de color de cada píxel asociado a esa localización (sistema YC_bC_r), pero ahora estamos ubicados en un dominio de frecuencias, donde se ha perdido esa asociación entre posición en la matriz y píxel de la imagen.

Hemos producido tres nuevas matrices de tamaño 8×8 (una para la luminancia y otras dos para las crominancias en cada bloque de la fotografía). En ellas, cada posición almacena una frecuencia diferente que, como ya hemos señalado, nos indica cómo de intensamente varía cada píxel con respecto a sus vecinos. Las bajas frecuencias se encuentran agrupadas en la zona superior izquierda de las matrices, mientras que las altas se localizan en la parte inferior derecha, (ver Figura (3.10)).

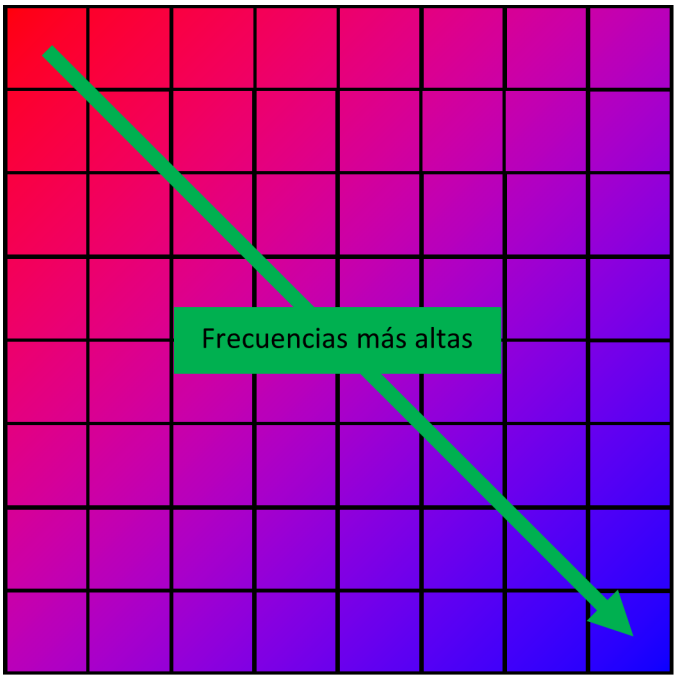


Figura 3.10: Esquema de frecuencias en un *MCU* tras aplicar la *2D DCT-II*.

Los coeficientes F_{uv} son números reales que no tienen por qué ser enteros. La siguiente modificación que experimenta cada *MCU* es convertir todos los números de la matriz en números enteros. Esto se logra con un proceso de cuantización. Desde el punto de vista matemático, es el paso más simple dentro de la compresión de la imagen, pero es en él donde

se produce la pérdida de información que permite comprimir más el archivo y es el causante de que no recuperemos el archivo en su estado original tras la compresión (ver sección 3.6). Ya hemos destacado antes que, como norma general, no se producen grandes cambios de color en un mismo *MCU* y, además, dichos cambios no son fácilmente perceptibles a simple vista, lo que va a facilitar la tarea de comprimir la imagen.

La cuantización depende principalmente de dos parámetros variables. El primero es el parámetro de cuantización α, $0 < \alpha < 1$, y representa la cantidad de compresión deseada. Cuanto mayor sea α, mayor será la compresión y, por tanto, menor será el tamaño de la imagen. No obstante, cuanto más comprimamos la imagen, menor será el parecido de la imagen descomprimida y la imagen original (ver Figura 3.11).

(a) $\alpha = 0.5$ (b) $\alpha = 0.95$ (c) $\alpha = 0.99$

Figura 3.11: Imagen de la catedral con diferentes parámetros de cuantización. Se aprecia cómo a medida que crece el valor de α, la imagen descomprimida presenta una calidad mucho inferior a la imagen original.

El segundo parámetro $[Q]$ se denomina matriz de cuantización, y es una matriz de tamaño 8×8 (almacenada en el propio archivo *.jpeg*) con valores predeterminados que han sido estudiados con el fin de minimizar la pérdida de calidad de la imagen y que se caracteriza por tener valores más grandes en la parte inferior derecha para eliminar las altas frecuencias. Como los cambios en la luminancia son más sensibles al ojo humano, existen diferentes matrices de cuantización para esta y para las crominancias. En (3.5) podemos ver unos valores típicos de las matrices de cuantización de la luminancia ($[Q_L]$) y de las crominancias ($[Q_C]$) recomendados por el estándar *JPEG* [16].

$$[Q_L] = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 131 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}, \quad [Q_C] = \begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}. \tag{3.5}$$

A partir de los coeficientes F_{uv} y los dos parámetros de cuantización se calculan los coeficientes cuantizados M_{uv} aplicando la fórmula

$$M_{uv} = signo\left(\frac{F_{uv}}{\alpha \cdot [Q]_{uv}}\right) \cdot \left\lfloor \left| \frac{F_{uv}}{\alpha \cdot [Q]_{uv}} \right| \right\rfloor, \tag{3.6}$$

De esta forma, si aplicamos (3.6) a los coeficientes de la matriz de luminancia (3.4) de nuestro ejemplo, con $\alpha = 0.75$ y $[Q_L]$ como en (3.5) como matriz de cuantización, obtenemos

$$[M] = \begin{pmatrix} 228 & 1 & 1 & 0 & 1 & -1 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.7}$$

El valor situado en la esquina superior izquierda es el coeficiente que aporta más información del bloque y se denomina *DC*. Al resto de valores se les denomina *AC* [3].

3.5. Proceso de codificación: el código Huffman

El proceso de cuantización que se ha realizado anteriormente tiene normalmente como resultado una secuencia de ceros en la parte inferior derecha de la matriz $[M]$. Conseguiremos facilitar el almacenamiento de los coeficientes cuantizados guardando la cantidad de ceros que hay al final, en lugar de cada uno de ellos. Para optimizar este proceso, leeremos la información de cada *MCU* siguiendo un patrón de zig-zag transversal, que podemos ver en la Figura 3.12.

En general, se codifican por separado y de forma diferente los valores *DC* y los *AC*. Esto se debe a que como hemos mencionado previamente, el término *DC* es el que tiene más cantidad de información. De esta manera, mientras que los *AC* se ordenan siguiendo el patrón de la Figura 3.12, los *DC* se codifican con las diferencias entre el *DC* del bloque que estamos mirando con el del bloque anterior. La razón de hacer esto es que existe una correlación muy elevada entre los elementos *DC* de bloques adyacentes.

En el ejemplo particular que estamos siguiendo durante este capítulo obtendremos la siguiente sucesión al leer la matriz de coeficientes cuantizados (3.7)

$$\begin{gathered} 228, 1, -3, -1, 0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, \\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. \end{gathered} \tag{3.8}$$

Ahora se agrupan las secuencias de ceros y se codifica la sucesión mediante una variante particular de la compresión *Run-Length Encoding* o *RLE*. Esta es una compresión de

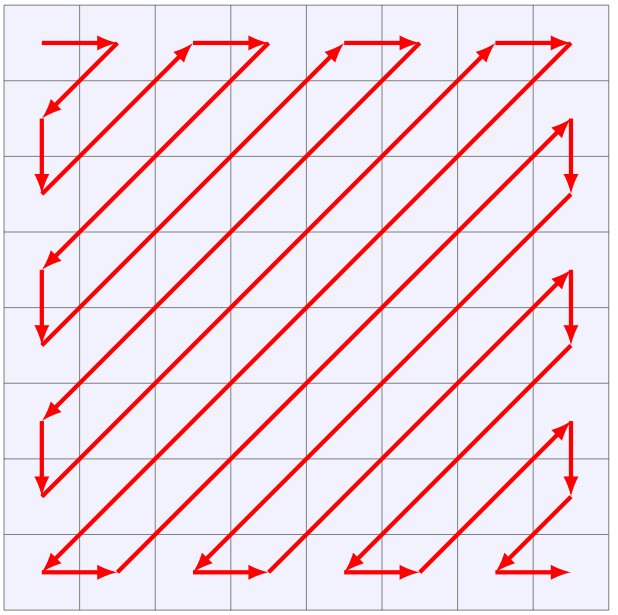


Figura 3.12: Patrón de zig-zag transversal.

datos sin pérdida que es muy útil para reducir el tamaño de sucesiones en las que hay repeticiones consecutivas de un mismo dato. Se sustituyen estas sucesiones por sucesiones de duplas formadas por el valor iterado y el número de veces que se ha repetido. Por ejemplo, la cadena $AAAAABBBBBBBBBBBAAACCCCCCCC$ se representaría por $[5, A], [11, B], [3, A], [7, C]$. De manera complementaria a la *RLE*, en algunos casos es útil aplicar otra técnica sencilla de compresión llamada *Delta Encoding*. Este método consiste en almacenar los incrementos en lugar de los valores directamente.

En la compresión del *JPEG* se codificarán los términos de la secuencia de la forma $[(r, s), c]$, donde:

- r: indica el número de ceros que preceden al coeficiente.
- s: señala el número de dígitos necesarios para codificar ese coeficiente.
- c: es el valor del coeficiente.

Por último, se utiliza el código $[(0, 0)]$ para indicar el final del bloque o lo que es lo mismo todo ceros hasta el final.

Ejecutamos la *RLE* al ejemplo (3.8) que estamos tratando y la secuencia queda codificada de esta manera:

$$[(0, 8), 228], [(0, 1), 1], [(0, 2), -3], [(0, 1), -1], [(1, 1), 1],$$
$$[(3, 1), -1], [(4, 1), 1], [(0, 1), -1], [(4, 1), -1], [(0, 0)]$$

Posteriormente, y como otra forma de comprimir más la secuencia se utiliza una versión específica de la codificación Huffman. Se basa en dos principios básicos:

1. Algunos grupos de valores se repiten mucho más que otros.

2. Los valores que aparecen más veces en la secuencia deberían necesitar menos bits para almacenarse.

La idea es asignar un código más corto a los valores que más se repiten.

La codificación Huffman es un algoritmo [12] cuya publicación en 1952 causó un gran avance en el ámbito de la compresión de datos. Su autor, David A. Huffman (1925–1999) demostró que utilizar árboles binarios de frecuencias ordenadas construidos de abajo hacia arriba era el código binario más eficiente. Vamos a ver un ejemplo para entender su funcionamiento.

Imaginemos que tenemos un texto, por ejemplo, el capítulo 21 del clásico de la literatura "El Principito" del escritor francés Antoine de Saint-Exupéry, y que queremos comprimir este texto en sistema binario de la forma más eficiente posible. Normalmente, codificaríamos cada carácter como un byte, es decir, un conjunto de 8 unos o ceros. Como el texto que estamos tratando tiene 6 093 caracteres, serían necesarios 48 744 bits para su codificación. Para aplicar la codificación Huffman, primero anotamos las frecuencias de cada carácter, o lo que es lo mismo, el número de veces que aparece dicho carácter en el texto (Tabla 3.1).

	1015	**e**	593	**o**	541	**a**	473	**s**	382	**i**	361	**r**	349	**n**	259
l	237	**d**	196	**t**	182	**c**	178	**u**	163	**.**	159	**m**	146	**p**	144
-	114	**\n**	67	**h**	54	**j**	52	**v**	50	**b**	48	**q**	48	**y**	47
z	46	**,**	46	**g**	46	**f**	21	**¿**	13	**?**	13	**¡**	13	**!**	13
:	10	**“**	4	**”**	4	**x**	2	**;**	1	**ñ**	1	**(**	1	**)**	1

Tabla 3.1: Frecuencias de cada carácter en el texto de "El Principito".

Ahora, empezando por aquellos con menor frecuencia, juntamos los caracteres que van a constituir las hojas de nuestro árbol binario. De esta forma, seleccionamos los dos caracteres con menor frecuencia de nuestra lista, que son ")" y "(", que aparecen únicamente una vez en todo el texto. Los juntamos poniendo ambos como nodos hijos de un padre cuya etiqueta será la suma de la frecuencia de los dos, y devolvemos esa etiqueta a la lista como si fuera un carácter más (Figura 3.13). Reiteramos el proceso hasta llegar al nodo raíz que, naturalmente, tendrá como etiqueta la suma de las frecuencias de todos los caracteres del texto.

Una vez que lo hayamos hecho y que hayamos obtenido el denominado árbol de Huffman, para conocer la codificación de cada carácter simplemente nos fijamos en el recorrido que tenemos que hacer en el árbol desde la raíz para llegar a la hoja correspondiente. De forma que según vayamos descendiendo por el árbol añadiremos un 0 al código si hemos bajado a la izquierda y un 1 si lo hemos hecho hacia la derecha. Por ejemplo, en el que caso que estamos tratando, los caracteres "r" y "." están identificados respectivamente como 0110 y 01001 (Figura 3.14).

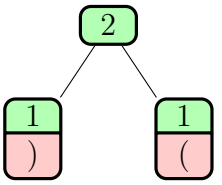


Figura 3.13: Construcción parcial de un árbol de Huffman.

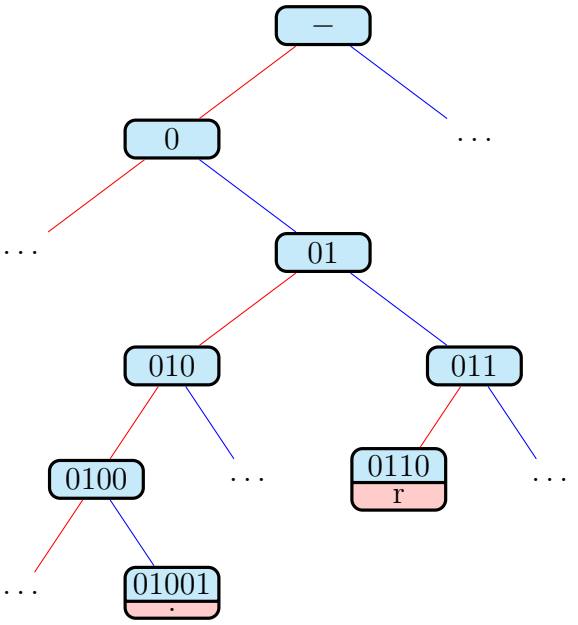


Figura 3.14: Codificación a partir de un árbol de Huffman.

En la Figura 3.15. podemos ver el árbol de Huffman completo construido a partir de la frecuencia de cada carácter en el fragmento que hemos escogido. A partir de este árbol, obtenemos el conjunto de codificaciones para cada carácter. En la Tabla 3.2 se puede ver la comparativa entre las que acabamos de obtener por el algoritmo de compresión de Huffman y las codificaciones tradicionales del código *ASCII*.

Como se puede apreciar en la tabla, los caracteres más utilizados necesitan bastante menos bits para codificarse que los menos utilizados. Esto permite que, a diferencia de la codificación *ASCII* donde todos los caracteres requieren el mismo número de bits, aquellos que aparecen de forma frecuente en el texto ocupen bastante menos espacio. De esta manera, por ejemplo, la letra “e”, que es el segundo carácter que más se repite en el texto (593 veces), se codifique con tres bits en comparación con los ocho de *ASCII*.

Aunque es cierto que la codificación de algunos de los caracteres menos recurrentes puede ser más larga que en *ASCII* (como la “ñ”, que usa trece caracteres), su baja frecuencia hace que el resultado total no se vea casi afectado. En resumen, los 48 744 bits necesarios en principio para almacenar este texto puede reducirse a 26 271 bits con el método de Huffman (aproximadamente, la mitad de espacio de almacenamiento).

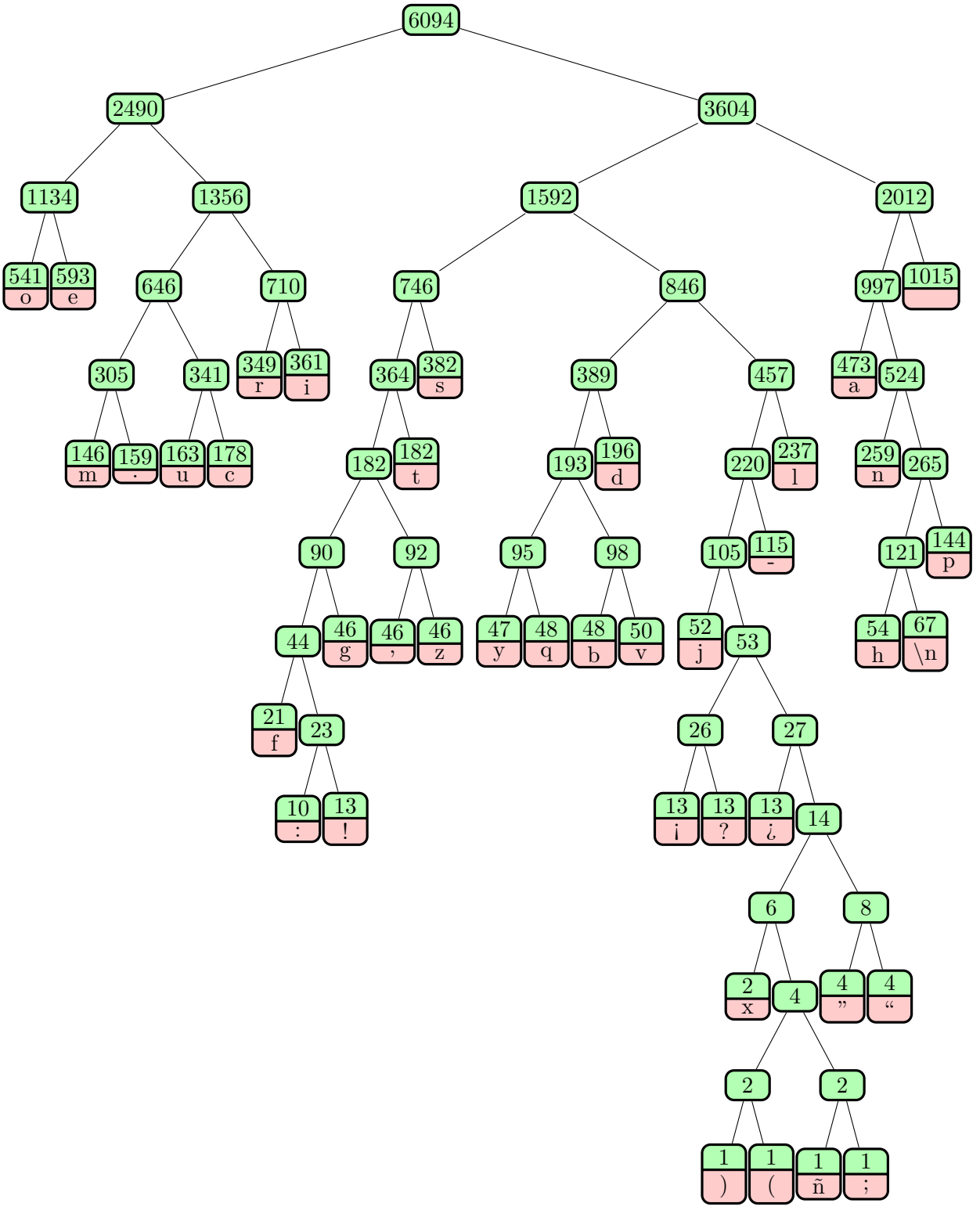


Figura 3.15: Árbol de Huffman de los caracteres del texto de “El Principito”.

	Frec.	***ASCII***	**Huffman**		**Frec.**	***ASCII***	**Huffman**
	1015	00100000	111	**e**	593	01100101	001
o	541	01101111	000	**a**	473	01100001	1100
s	382	01110011	1001	**i**	361	01101001	0111
r	349	01110010	0110	**n**	259	01101110	11010
l	237	01101100	10111	**d**	196	01100100	10101
t	182	01110100	10001	**c**	178	01100011	01011
u	163	01110101	01010	**.**	159	00101110	01001
m	146	01101101	01000	**p**	144	01110001	110111
-	114	00101101	101101	**\n**	67	00001010	1101101
h	54	01101000	1101100	**j**	52	01101010	1011000
v	50	01110110	1010011	**b**	48	01100010	1010010
q	48	01110001	1010001	**y**	47	01111001	1010000
z	46	01111010	1000011	**,**	46	00101100	1000010
g	46	01100111	1000001	**f**	21	01100110	10000000
¿	13	10111111	101100110	**?**	13	00111111	101100101
¡	13	10100001	101100100	**!**	13	00100001	100000011
:	10	00111010	100000010	**“**	4	00100010	10110011111
”	4	00100010	10110011110	**x**	2	01111000	10110011100
;	1	00111011	1011001110111	**ñ**	1	11110001	1011001110110
(	1	00101000	1011001110101	**)**	1	00101001	1011001110100

Tabla 3.2: Comparativa de las codificaciones *ASCII* y método Huffman para los caracteres del texto de “El Principito”.

3.6. Procesos inversos

Por último, junto a todo método de compresión es imprescindible tener un algoritmo de descompresión para recuperar la imagen original. Vamos a explicar cómo invertir el proceso visto en las secciones anteriores a la hora de reconstruir una imagen comprimida. Un detalle importante que no hemos mencionado explícitamente hasta ahora es que todas las operaciones realizadas en el proceso de compresión se pueden invertir. Esto ya podía intuirse de nuestro estudio en el capítulo 2 de la *DFT* y las *DCT*.

Para deshacer la codificación Huffman, únicamente atenderemos al árbol de Huffman de nuestra imagen (Figura 3.15). Podemos decodificar cada carácter si descendemos desde la raíz del mismo (hacia la izquierda si tenemos un 0 o hacia la derecha si es un 1) hasta llegar a una hoja, tal y como vimos en la Figura 3.14. Posteriormente, deshacemos la técnica *RLE* escribiendo los ceros entre las sucesiones de coeficientes correspondientes. Tras haber

obtenido la sucesión completa, podemos "recuperar" la matriz de coeficientes cuantizados si ordenamos los términos siguiendo la estructura de *zig-zag* transversal vista en la Figura 3.12.

A partir de la matriz M_{uv} aplicamos la fórmula

$$F_{uv} = signo\,(\alpha \cdot [Q_{uv}] \cdot M_{uv}) \cdot \lfloor |\alpha \cdot [Q_{uv}] \cdot M_{uv}| \rfloor$$

para "recuperar" los coeficientes F_{uv} del espacio de color YC_bC_r.

Basándonos en (2.30), definimos la transformada discreta de coseno inversa bidimensional *2D ICDT-II* mediante la expresión

$$\frac{2}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} Q_u Q_v G_{uv} \cos\left(\frac{(2x+1)u\pi}{2M}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right),$$

donde $x \in \mathbb{Z}_M$, $y \in \mathbb{Z}_N$ y Q_k está definido en (2.22). Para "recuperar" las matrices $[Y]$, $[C_b]$ y $[C_r]$ deshacemos la *2D DCT-II* con

$$f_{xy} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} Q_u Q_v F_{uv} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right), \qquad x, y \in \mathbb{Z}_8.$$

El cambio al sistema de color *RGB* se logra invirtiendo la transformación (3.1):

$$\begin{pmatrix} 0,29900 & 0,58700 & 0,11400 \\ -0,16874 & -0,33126 & 0,50000 \\ 0,50000 & -0,41869 & -0,08131 \end{pmatrix}^{-1} = \begin{pmatrix} 1,00000 & 0,00001 & 1,40200 \\ 1,00000 & -0,34413 & -0,71414 \\ 1,00000 & 1,77200 & 0,00002 \end{pmatrix}.$$

Es decir, la transformación inversa es

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1,00000 & 0,00001 & 1,40200 \\ 1,00000 & -0,34413 & -0,71414 \\ 1,00000 & 1,77200 & 0,00002 \end{pmatrix} \begin{pmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{pmatrix}.$$

Finalmente, se unen todos los bloques *MCU*, para formar la imagen descomprimida.

Capítulo 4

Conclusiones

En este trabajo hemos abordado de manera exhaustiva cada una de las etapas involucradas en el estándar de compresión y codificación de imágenes del *JPEG* (descomposición en bloques y tratamiento del color, transformada discreta de coseno, proceso de cuantización y proceso de codificación), realizando un análisis detallado de los fundamentos matemáticos necesarios presentes en el proceso. Hemos podido explorar conceptos clave del análisis de Fourier, área que desempeña un papel fundamental en diversos campos y disciplinas, destacando la importancia de la transformada discreta de coseno y su eficiente cálculo mediante algoritmos específicos. Además, se han descrito en detalle las técnicas de compresión y codificación utilizadas en el proceso del *JPEG*, desde la cuantización de los coeficientes hasta la aplicación de algoritmos como zig-zag transversal, *Run-Length Encoding*, *Delta Encoding* y codificación Huffman.

La elaboración de este estudio ha representado un proceso desafiante y enriquecedor en el campo de la compresión y codificación de imágenes. Durante décadas, el estándar del *JPEG* ha sido ampliamente utilizado para la transmisión y almacenamiento de imágenes digitales, por lo que resulta fundamental comprender las matemáticas subyacentes a este algoritmo. Aunque este trabajo se ha centrado en el proceso *baseline sequential process*, existen otras variantes y técnicas más avanzadas que podrían ser exploradas en futuras investigaciones. Una de las cosas que he podido comprender y apreciar es la complejidad matemática involucrada en un estándar de compresión, así como su relevancia en la transmisión y almacenamiento eficiente de imágenes digitales.

En conclusión, este trabajo ha sido una valiosa oportunidad para adentrarse en las matemáticas detrás del estándar de compresión y codificación de imágenes del *JPEG*. A través del estudio detallado de las diferentes etapas del algoritmo, se ha obtenido un conocimiento sólido y una comprensión más profunda de cómo se logra la compresión y codificación eficiente de imágenes. En lo personal, este aprendizaje no ha sido relevante únicamente en el ámbito académico, sino también en el contexto de la tecnología digital que influye en nuestra vida cotidiana.

Bibliografía

[1] N. Ahmed. How I Came Up with the Discrete Cosine Transform. *Digital Signal Processing*, 1:4–5, 1991.

[2] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete Cosine Transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974.

[3] D. Austin. Image Compression: Seeing What's Not There. *Notices of the AMS*, 2007.

[4] D. Austin. What is... JPEG? *Notices of the AMS*, 55(2):226–229, 2008.

[5] J. F. Blinn. What's the Deal with the DCT? *IEEE Computer Graphics and Applications*, 13(4):78–83, 1993.

[6] W-H. Chen, C. Harrison Smith, and S. C. Fralick. A Fast Computational Algorithm for the Discrete Cosine Transform. *IEEE Transactions on Communications*, 25(9):1004–1009, 1977.

[7] J. W. Cooley, P. A. W. Lewis, and P. D. Welch. Historical Notes on the Fast Fourier Transform. *Proceedings of the IEEE*, 55(10):1675–1677, 1967.

[8] J. W. Cooley and J. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[9] C. J. Van der Branden Lambrecht. *Vision Models and Applications to Image and Video Processing*. Kluwer Academic Publishers, 2001.

[10] J. Duoandikoetxea. *Fourier Analysis*. American Mathematical Society, 2001.

[11] G. Hudson and A. Léger and B. Niss and I. Sebestyén. JPEG at 25: Still Going Strong. *IEEE MultiMedia*, 24(2):96–103, 2017.

[12] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[13] D. E. Newland. *Vibraciones aleatorias y análisis espectral*. AC, Madrid, 1983.

[14] K. R. Rao and P. Yip. *Discrete Cosine Transform: Algorithms, Advantages Applications.* Academic Press, 1990.

[15] G. Strang. The Discrete Cosine Transform. *SIAM Review*, 41(1):135–147, 1999.

[16] The International Telegraph and Telephone Consultative Committee (CCITT). Terminal equipment and protocols for telematic services: Information technology-digital compression and coding of continuous-tone still images-requirements and guideline. *ISO/IEC International Standard 10918-1. Recommendation T.81*, 1992.

[17] K. S. Thyagarajan. *Still Image and Video Compression with MATLAB.* Wiley John and Sons, 2010.