UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Un ejemplo visual del uso de la DCT en el estándar JPEG mediante OpenCV

Apellidos, nombre	**Agustí i Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Departamento de Informática de Sistemas y Computadores**
Centro	**Escola Tècnica Superior d'Enginyeria Informàtica** Universitat Politècnica de València

1 Resumen

Cuando se habla de representación de imágenes en computadores es habitual hablar de su representación en modo “mapa de bits”. Esta representación se realiza en el **dominio del espacio** y consiste en la digitalización de la intensidad de la luz que el sensor de una cámara (o un escáner) ha obtenido de una escena del mundo real, véase[1] la Figura 1a (en este caso en niveles de gris, por simplicidad). Esta representación obtiene un conjunto de valores numéricos organizados en forma de matriz, donde cada elemento es la información numérica asociada a cada punto de esa conversión analógico a digital. El uso de cámaras digitales, televisores y, sobre todo, de teléfonos móviles con cámara ha hecho popular el término de megapíxeles y lo utilizamos como un parámetro global de calidad del dispositivo utilizado y de la imagen obtenida. La Figura 1b muestra cómo el cine ha hecho popular algunas de estas representaciones al estilo del “Arte Ascii”[2], donde la luminosidad de la imagen se representa con caracteres de diferente nivel de brillo.

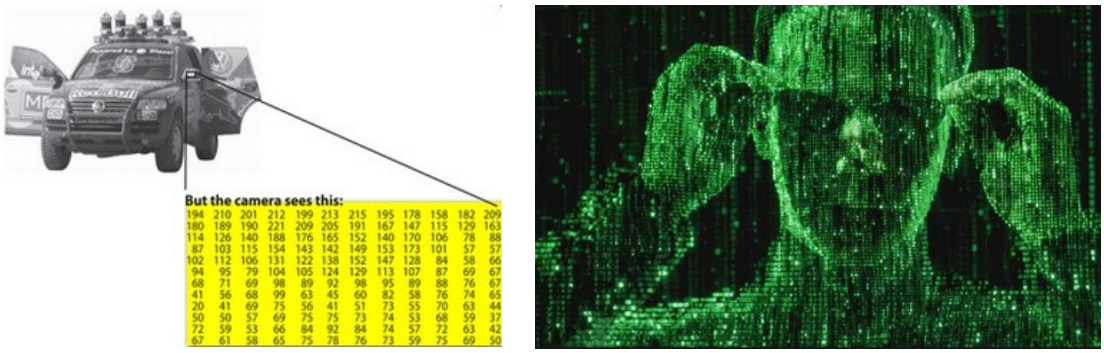


Figura 1: Representación gráfica de la imagen en un computador: como abstracción formal (a) y como ficción cinematográfica (b).

Es usual también utilizar nombres de formatos de imágenes (como JPEG [1]) y asociar a este formato con una representación de imágenes que es capaz de obtener un tamaño de fichero muy bajo al ser capaz de “comprimirlas con mucha calidad”. Esta característica que se puede oír (y leer) en muchas ocasiones se debe a que en el modo de funcionamiento de JPEG está contemplado el uso de la compresión y, especialmente, se ha hecho popular el resultado de su **compresión con pérdidas** (*loosy* en la terminología anglosajona). A diferencia de la representación interna de los valores de una imagen basados en niveles de luminosidad, el grueso del funcionamiento interno de JPEG representa una imagen en el **domino frecuencial**: basado en el uso de la distribución de frecuencias de los valores de intensidad de luz mediante el algoritmo DCT[3].

---

1 Imágenes obtenidas de "OpenCV: Mat - The Basic Image Container" <https://docs.opencv.org/3.2.0/d6/d6d/tutorial_mat_the_basic_image_container.html> y “¿Cómo debería ser nuestro universo si estuviéramos en Matrix?” <https://www.taringa.net/+ciencia_educacion/matrix_12vfvk>, respectivamente.

2 Véase el artículo sobre *ASCII art* al respecto en la *Wikipedia*. Disponible en <https://en.wikipedia.org/wiki/ASCII_art>.

3 Véase el artículo sobre DCT al respecto en la *Wikipedia*. Disponible en <https://en.wikipedia.org/wiki/Discrete_cosine_transform>.

Esta no es una representación obvia y en este artículo voy a mostrar, de manera visual, cómo es y cómo están relacionadas las frecuencias y las "pérdidas" de la compresión JPEG.

2 Objetivos

Una vez que el lector haya explorado los contenidos de este documento, será capaz de:

- Relacionar el contenido de una imagen con valores de frecuencias.
- Experimentar con un ejemplo de código que permite visualizar la representación frecuencial que utiliza JPEG y cómo los valores de esta representan la distribución espacial de los niveles de intensidad.
- Experimentar, con un ejemplo de código para visualizarlo, cómo la repetida aplicación de la compresión con pérdidas puede llegar a afectar a la imagen representada.

Para hacer posible esta experimentación, sin entrar en los detalles de todo el proceso de cómo JPEG utiliza el dominio de las frecuencias para representar el contenido de las imágenes, voy a utilizar la librería OpenCV [2]. Esto me permitirá mostrarlo de manera visual y observar cómo consigue la compresión reducir el tamaño del fichero a costa de modificar la representación del contenido de la imagen.

Aunque puedes leer este documento y ver si te convencen las imágenes, la diversión está en probar los ejemplos y modificarlos, será la mejor manera de aprender de ellos. Anímate y ve preparando un terminal y un editor de código.

Este trabajo se centrará en la plataforma *Linux/Unix*, pero existen las mismas herramientas o equivalentes en otros sistemas operativos. De hecho, el haber optado por OpenCV para desarrollar el código del ejemplo es para que sea totalmente portable.

3 Introducción

Existen muchos formatos de ficheros para la representación de imágenes y se pueden encontrar comparativas entre ellos ([3] y [4]) para ayudar a tomar la decisión de cuál escoger. De entre ellos JPEG [1] tiene un gran uso por su capacidad de reducir el tamaño del archivo manteniendo la fidelidad visual, esto quiere decir que quien ve la imagen comprimida junto al original puede apreciar la mayor parte de los detalles de la misma.

Interiormente [5] emplea el algoritmo DCT para obtener una descripción de la imagen basada en frecuencias y una etapa de cuantización en la que el usuario puede especificar el nivel de fidelidad que quiere mantener. Hay un bloque más, *Entropy Encoder*, que es un paso en que, de forma estadística, se busca reducir las redundancias de los valores que aparecen repetidos en la imagen. Este no voy a entrar a verlo porque es una cuestión matemática que también emplean los métodos de compresión sin pérdidas (*loossless*) y, por lo tanto, no es tan distintiva de este tipo de compresión característico de JPEG.

Son pues las salidas de los bloques *FDCT* y *Quantizer* que muestra la Figura 2a los que vamos a explorar y modificar para ver qué información se descarta o se mantiene. La influencia de este bloque es lo que se quiere mostrar, sin entrar en los términos matemáticos que lo describen. El proceso es reversible afortunadamente. ¿Te imaginas qué pasaría si no fuera así? La Figura 2b muestra el paso de la descompresión, que sigue a la lectura del fichero en disco para volver a obtener una imagen.

Decimos mantener porque, en la compresión con pérdidas que estamos analizando, **insisto**, parte de la información de la imagen original se descarta, se pierde. Así que lo que se lee de un fichero JPEG tiene algunas diferencias con la imagen de partida que se utilizó para generarlo.

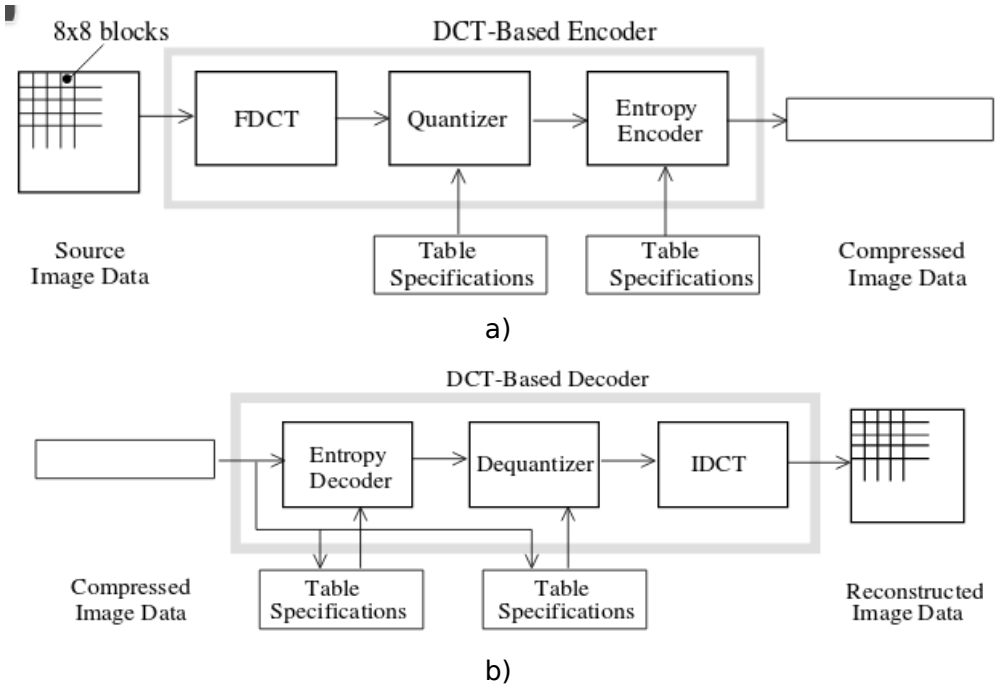


Figura 2: The JPEG Still Picture Compression Standard [5]: el codificador (a) y el decodificador (b).

¿Cómo lo vamos a ver? Vamos llevar a cabo este proceso de generación de un fichero JPEG utilizando las operación de manipulación de imágenes y ficheros de la librería OpenCV. En el camino para generar el fichero modificaremos los resultados intermedios de forma gradual y mostraremos el resultado para que se vea cómo influye la capacidad del usuario de especificar un nivel de pérdidas o de fidelidad visual en el proceso. Analizaremos el tamaño del fichero obtenido en cada resultado intermedio para observar el impacto en el espacio que se necesita para almacenar la imagen. Y lo compararemos con un esquema como PNG que es un buen representante de esquema de compresión sin pérdidas.

Aunque JPEG se aplica a cada canal de la imagen en color (RGB, o rojo verde y azul), trabajaremos sobre la versión de grises de la imagen original que solo tiene un canal y evitaremos la redundancia de operaciones en el código que te vamos a mostrar, así como las operaciones necesarias de separación y mezcla posterior de canales. Te lo dejo como experimento para ti, estimado lector.

4 Desarrollo

Para verlo vamos a utilizar un ejemplo que hace parte de lo que queremos, esto es, nos permite recorrer las etapas principales del proceso de generación de un JPEG conectando las dos partes que muestra la Figura 2): genera la imagen de frecuencias, la modifica y reconstruye la imagen con esa versión modificada. Respectivamente, son los pasos denominados *FDCT*, *Quantizer* e *IDCT* en la Figura 2. Sobre este código de partida voy a proponer qué cosas se pueden añadir para permitir al usuario explorar la representación en frecuencias.

4.1 Ejemplo de partida

El trabajo de [2] hace parte de lo que queremos: genera cien posibles ficheros que corresponden con una eliminación (un borrado) incremental de las frecuencias que se han calculado en las imágenes y, por tanto, determina las "pérdidas" que se provocan en el resultado final. Este ejemplo no proporciona ninguna salida gráfica en pantalla, así que no es inmediato ver un resultado.

Figura 3: Resultados de la compresión JPEG.

La Figura 3 muestra nueve imágenes del total, las de mayor número tienen más fidelidad con la original y las dos últimas (especialmente la 0010) es muy notable que se ha perdido gran parte de la información. Recuerde es por utilizar la versión de frecuencias de la imagen en la que se han borrado **intencionadamente** algunos valores. Así que es posible ver las pérdidas al revertir el proceso y comparar con la imagen de partida.

El código lo muestra el Listado 1, con los comentarios originales, pero cambiando el formato de las rutas para hacerlo portable. A destacar del mismo cuál es papel de las variables que hacen referencia a las imágenes que son las etapas anteriores del proceso:

1. Codificación
   a) Tras la lectura de la imagen (*img*), para aplicar la transformación en frecuencias redimensiona la misma si es necesario (*img2*) y, sobre esta, recodifica sus valores de enteros a reales (*img3*).
   b) Obtiene la *FDCT* (*freq*) con la función *dct* de OpenCV y modifica los valores de esta matriz a modo de algoritmo simple de cuantización (*Quantizer*), en el bucle haciendo variar *Start* desde 100 hasta 0, eliminando frecuencias contiguas, empezando por las altas y así permitiendo ver de forma creciente el impacto sobre la calidad de la imagen y sobre el tamaño del fichero.
2. Decodificación

   A partir de esta matriz de frecuencias (*freq*) modificada (que sería como haber leído una imagen JPG desde fichero, aplica la inversa de la *FDCT*, la *IDCT*, para obtener la imagen reconstruida (*dst)*.

Para compilarlo y ejecutarlo se puede hacer con

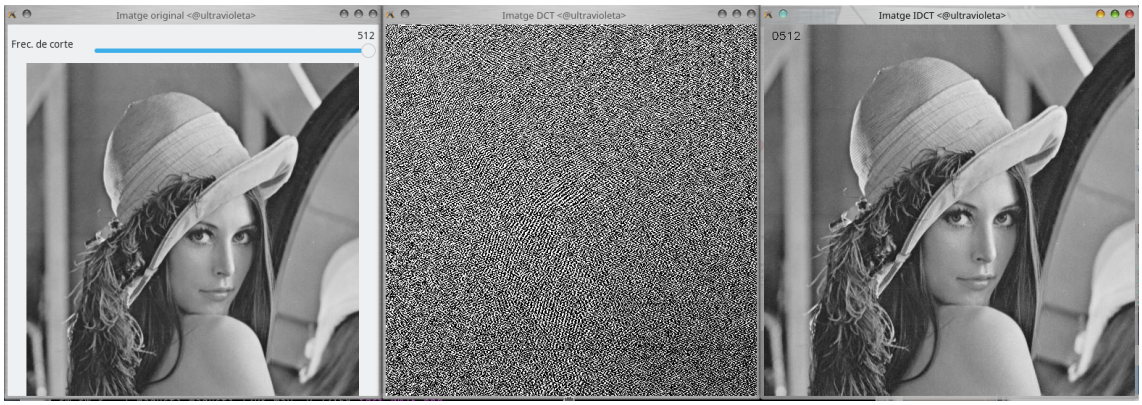
```
$ g++ codic.cpp -o codic `pkg-config openal opencv --libs` -lm
$ codic lena.jpg
```

```
// Read image
Mat img = imread(argv[0], CV_LOAD_IMAGE_GRAYSCALE);
// Make sure the both image dimensions are a multiple of 2
Mat img2, img3, freq, dst;
int w = img.cols, h = img.rows, w2,h2;
if (w % 2 == 0)     w2 = w; else     w2 = w+1;
if (h % 2 == 0)     h2 = h; else     h2 = h+1;
copyMakeBorder(img, img2, 0, h2-h, 0, w2-w, IPL_BORDER_REPLICATE);

// Grayscale image is 8bits per pixel,
// but dct() method wants float values!
img3 = Mat( img2.rows, img2.cols, CV_64F);
img2.convertTo(img3, CV_64F);
imwrite("orig.png", img3);
// Let's do the DCT now: image => frequencies
dct(img3, freq);
// Save a visualization of the DCT coefficients
imwrite("dct.png", freq);

for (int Start=100; Start>0; Start-=1) {
    // Set some frequencies to 0
    for (int y=Start; y<freq.rows; y++)     {
        for (int x=Start; x<freq.cols; x++)         {
            freq.at<double>(y,x) = 0.0;
        }
    }
    // Do inverse DCT: (some low) frequencies => image
    idct(freq, dst);
    // Show frame nr
    char txt[100];
    sprintf(txt, "%04d", Start);
    cv::putText( dst, txt, Point(10,20), CV_FONT_HERSHEY_SIMPLEX,
                 0.5, CV_RGB(0,0,0) );
    // Save visualization of reconstructed image where we have thrown
    // away some of the high frequencies
    char fname[500];
    sprintf(fname, "idct_%04d.png", Start);
    imwrite(fname, dst);
}
```

Listado 1: Código básico: ejemplo de partida, extraído de [2].

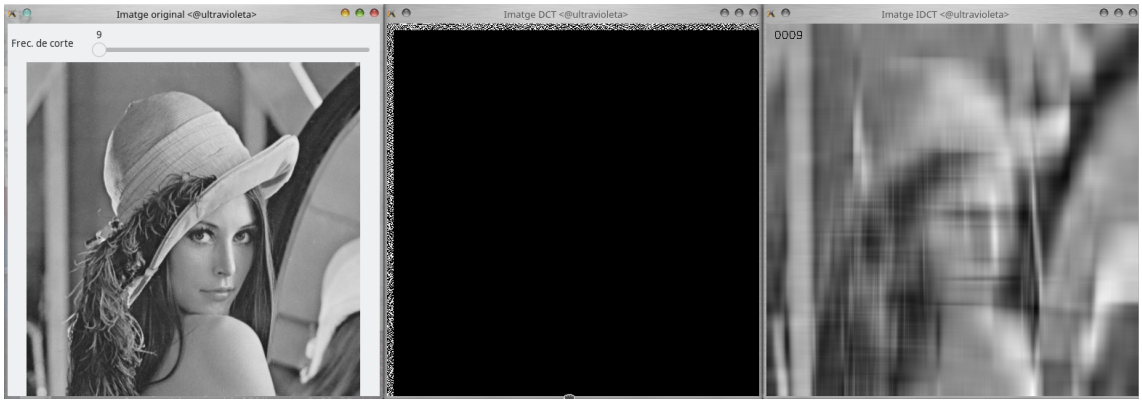
4.2 Visualizar la secuencia de pasos

Para ver lo que está pasando se ha modificado el código original. La Figura 4 muestra lo que sería el inicio de la aplicación: a la izquierda la imagen original y a la derecha la imagen reconstruida. En el centro, la imagen de frecuencias muestra valores en sus puntos que corresponden a si está o no cada frecuencia en la imagen: desde las más bajas (arriba a la izquierda) a las más altas (abajo a la derecha). Si no se modifica la representación en frecuencias, el resultado obtenido es igual a contenido de partida.



a) b) c)

Figura 4: Ventanas de la nueva interfaz de la aplicación: imagen original (1), representación en frecuencias (b) e imagen reconstruida (c).



a) b) c)

Figura 5: Resultados obtenidos con la frecuencia de corte a valor 9.

Si modificamos la imagen de frecuencias, el resultado reconstruido ya no será el original. La Figura 5 muestra un caso extremo en el que se han puesto a cero todas las componentes de frecuencias por debajo de las fila y columna siete. ¿Ves el cuadrado negro que hay en la imagen del centro? ¡Pues eso es!. Al utilizarlo como partida para revertir el proceso se obtiene una imagen en la que todavía hay mucha de la información de partida pero, creo que estarámos de acuerdo en que no sería la que escogeríamos ... Al menos no en la mayor parte de usos de esta imagen.

La graduación que mostraba la Figura 3 se ha conseguido haciendo variar el tamaño de ese rango de componentes de frecuencias que son borradas: la aplicación las ha puesto a cero. De una manera muy simplista, muy binaria, esta es la idea del esquema de compresión de JPEG: el usuario propone un nivel de fidelidad de la imagen a mantener y eso, es la base para que el algoritmo escoja qué valores se mantienen o se hacen cero de ese rango. A menor dimensión de este, menor compresión y viceversa.

Para verlo de una forma interactiva, la barra de desplazamiento, en la parte superior de la imagen original, permite especificar este valor de dimensión del cuadrado de recorte de frecuencias. Para observarlo de una manera secuenciada he hecho vídeos, tomando cada cuadro del mismo con un valor diferente de frecuencia de corte. Pero no los puedo mostrar aquí en el papel.

4.3 ¿Que esconde la imagen de frecuencias?

La imagen de partida que estamos utilizando tiene de casi todas las frecuencias, como es habitual en una fotografía. En el caso de dibujos esta variedad no suele ser tan amplia. Para observar cómo en la matriz de frecuencias se representan las frecuencias orientadas espacialmente he utilizado una imagen con un contenido muy geométrico: un tablero de ajedrez, aunque solo de 6x5 elementos.

La imagen superior de la Figura 6 muestra la imagen, su representación frecuencial y la reconstrucción. ¿A que observas un patrón de repetición en los puntos de esta? ¡Lo has visto! La imagen se repite a valores múltiplo del ancho de un cuadro negro y uno blanco en horizontal. ¡Y también en vertical, no lo olvidemos!

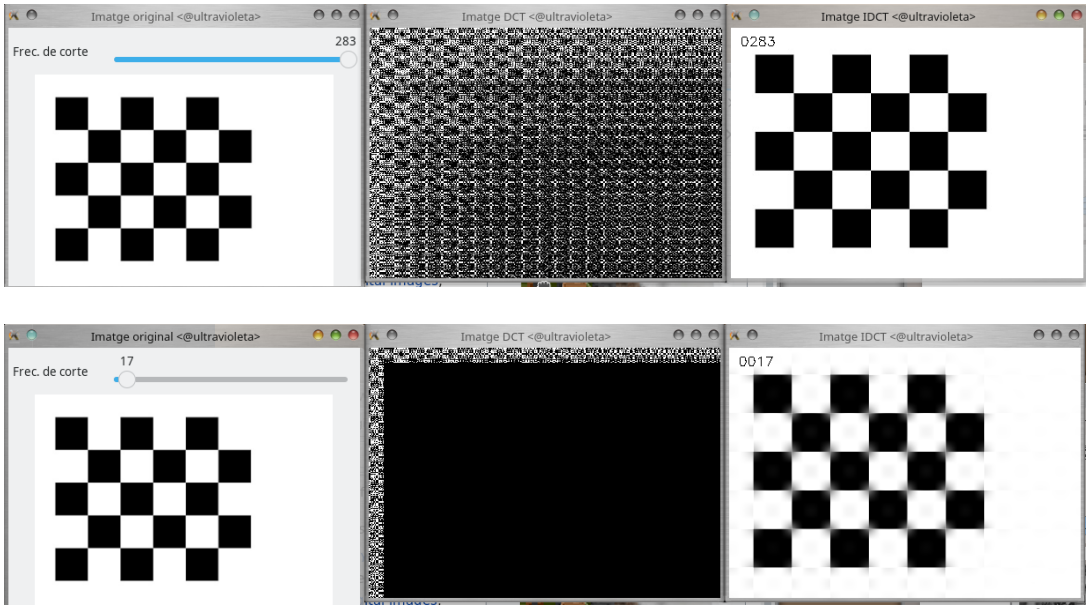


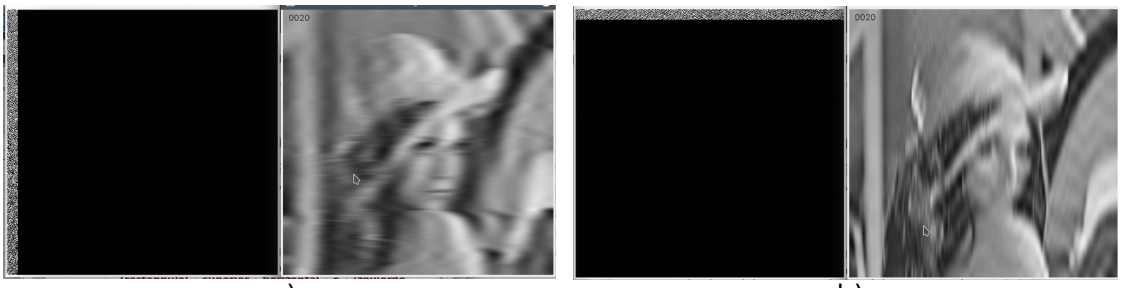
Figura 6: Ejemplo con una imagen generada para obtener un contenido más geométrico y simple del que es habitual en una fotografía.

Si, como veníamos haciendo, modificamos la descomposición frecuencial y reconstruimos (como en la fila de abajo de la Figura 6), se observará que la imagen reconstruida empieza a mostrar artefactos en las direcciones diagonales, debido a la eliminación de información de frecuencias introducida.

Aplicado eso a la imagen original podemos ver esa direccionalidad utilizando un valor muy bajo de la frecuencia de corte (20, para que sea fácilmente visible la eliminación de los valores de frecuencia) y modificando el código original para que la zona de recorte ahora no sea un cuadrado, sino un rectángulo y así se puede borrar selectivamente una franja horizontal o vertical para mostrar el resultado.

La Figura 7 muestra los resultados obtenidos: ahora el recorte de frecuencias se aplica a toda la matriz en una dimensión, pero a la otra solo a partir de donde se especifica el valor de corte. Observe la imagen de frecuencias de la Figura 7a y verá que hemos dejado una franja vertical de los valores originales, eso quiere decir que se conservan todas las frecuencias bajas en sentido horizontal, pero muy poco en vertical. Mientras que en la Figura 7b se mantienen parte de las

frecuencias verticales, así que los elmentos de fondo que están en esa orientación aparecen bastante bien perfilados. Se observa que, en ambos casos, en la imagen reconstruida, se introducen artefactos (emborronamientos) en la dirección en la que se han borrado los coeficientes de frecuencias.



a) b)

Figura 7: Efecto de un recorte orientado de las componentes de frecuencia: en vertical (a) y horizontal (b).

Esta direccionalidad es debida a que la DTC que estamos empleando es bidimensional y se calcula como una combinación de dos DCT unidimensionales. La Figura 8 muestra una representación visual habitual de la DCT en la que se puede ver esa combinación y orientación y que coincide que lo que acabamos de ver en la Figura 7

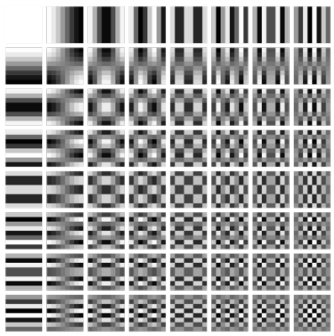


Figura 8: Representación visual de la DCT-2D como combinación de dos DCT-1D horizontal (en filas) y vertical (en columnas). Imagen extraída de [6]

5 Conclusión

Hemos visto cómo se modifica el contenido de una imagen al aplicar un esquema de codificación y compresión de las imágenes basado en una representación frecuencial, en lugar de una representación espacial de los valores de luminancia que obtienen los sensores de las cámaras.

Si está interesado en el código completo para visualizar las etapas intermedias y explorar con los valores no tiene más que enviarme un correo electrónico y se lo haré llegar. No quiero aburrir con los detalles de la implementación. Con él podrá comprobar, con otras imágenes, po. ej., con imágenes de edificios que tienen una buena cantidad de lineas, la direccionalidad de la imagen de frecuencias.

Solo quiero destacar dos detalles para acabar. Por un lado la variación del tamaño que ofrece JPG como representante de un esquema de compresión con pérdidas, frente a uno sin pérdidas como PNG. La Figura 9 muestra esta de forma gráfica: la misma imagen de partida, en PNG baja ocupa desde 156219 hasta los 91142 bytes y JPG desde 49394 hasta 19382 bytes.

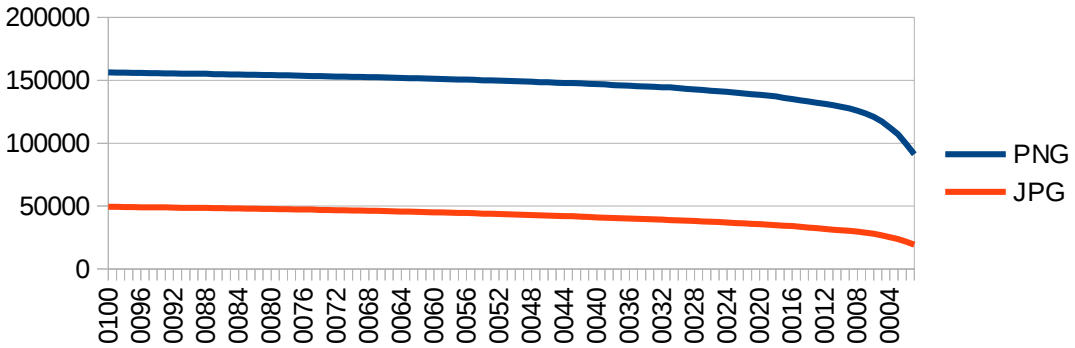


Figura 9: Evolución del tamaño de la imagen en JPG vs PNG.

Ah, y si cojo un fichero JPEG y lo vuelvo a grabar con compresión, con pérdidas y lo vuelvo a hacer con el resultado obtenido y así muchas veces, e resultado será … ¿Lo adivinas? Lo verás en la Figura 10. En el caso de PNG (como es un esquema de compresión sin pérdidas) … Lo verás en la misma figura.

Figura 10: Efecto de recodificar una imagen: PNG vs JPG.

6 Bibliografía

[1] Joint Photographic Experts Group (JPEG). Disponible en <https://jpeg.org/>.

[2] K. Toennies . (2012). DCT, DFT, and DST. Disponible en <http://www.juergenwiki.de/old_wiki/doku.php?id=public:dft_dct_dst>.

[2] OpenCV (Open Source Computer Vision Library). Disponible en <https://opencv.org/>.

[3] Solved - Best Image Format for the Web? PNG, JPG, GIF, and SVG. Philip Westfall, January 23, 2018 Disponivle en <https://www.pagecloud.com/blog/web-images-png-vs-jpg-vs-gif-vs-svg>.

[4] William Craig. JPEG 101: A Crash Course Guide on JPEG. WebFX, Inc. Disponible en <https://www.webfx.com/blog/web-design/jpeg-101-a-crash-course-guide-on-jpeg/ >.

[5] Gregory K. Wallace. (1991). The JPEG Still Picture Compression Standard. Multimedia EngineeringDigital Equipment Corporation. Maynard,. Massachusetts IEEE Transactions on Consumer Electronics.

[6] Wikipedia contributors. Discrete cosine transform. In *Wikipedia, The Free Encyclopedia*. Disponible en <https://en.wikipedia.org/w/index.php?title=Discrete_cosine_transform&oldid=895334325>.