Compresión de imágenes mediante DCT Quantization

Martín Sturla, Darío Sneidermanis
Estudiantes Instituto Tecnológico de Buenos Aires (ITBA)

31 de Mayo de 2012

(Paper No Publicado)

***Resumen*—El siguiente paper busca analizar la compresión de imágenes utilizando *DCT Quantization* según la especificación JPEG original, comparando el nivel de compresión alcanzado y calidad perdida utilizando distintos niveles de cuantización.**

***Palabras clave*—JPEG, compresión de imágenes, transformada discreta coseno, cuantización.**

I. Introducción

Históricamente la compresión de datos siempre ha sido de notable interés para la informática, ya sea para guardar algún archivo en algún dispositivo de capacidad limitada o para enviar un archivo por algún enlace utilizando la menor cantidad de banda ancha posible. Debido a estos requerimientos, surgen los primeros programas que haciendo uso de técnicas de compresión de datos, por ejemplo *Run Length Encoding* o codificación de *Huffman*, comprimen y descomprimen archivos. Algunos ejemplos incluyen *Winzip*, lanzado en 1991, o *Winrar*, lanzado en 1993. Sin embargo en el caso de las imágenes, debido a la naturaleza de sus datos, existen mejores alternativas para comprimirlas. Ya en el 1992, un comité conocido como *Join Photographic Experts Group* comienza a publicar pautas para la compresión de imágenes, conocidas como la especificación *JPEG*.

La especificación original de JPEG, en la cual se centra este trabajo, indica que la compresión de las imágenes consiste en dos etapas fundamentales: una codificación con pérdida de información, es decir *lossy*, seguida de una compresión *lossless*, como por ejemplo las dos ya mencionadas. En esta primer etapa existen dos maneras de reducir la información: *chroma subsampling* y *DCT Quantization*. La primera, la cual no es analizada en este trabajo, consiste en aprovechar que el ojo humano tiene una menor agudeza para diferenciar colores que brillantez o *luma*, por lo cual se reduce la resolución de la croma. La segunda consiste en hacer uso de que el ojo humano es capaz de diferenciar cambios de luma en zonas grandes, pero no tan tenazmente en zonas pequeñas. Debido a esto, se eliminan estas componentes de frecuencia espaciales altas en zonas pequeñas (casi imperceptibles para el ojo) y se suavizan los cambios de luma (también se efectúa un proceso similar con la croma). Debido a esta eliminación de información, existe mayor redundancia de los datos, por lo que un llamado posterior a un algoritmo de compresión será más efectivo.

El objetivo primordial del trabajo es analizar la técnica de *DCT Quantization* para distintas imágenes y matrices de cuantización, analizando en cada caso la tasa de compresión, el error introducido a la imagen recuperada y su distorsión gráfica. La sección 2 explica en profundidad la primer etapa de la compresión. La sección 3 trata sobre la segunda etapa de compresión. La sección 3 explica brevemente cómo recuperar la imagen. La sección 4 contiene los resultados.

II. Etapa de compresión con pérdida de información

A. Transformación del espacio de colores

La primer etapa de compresión se centra en el manejo de luma y croma, por lo cual es necesario transformar los datos de las imágenes, que normalmente contienen la cantidad de rojo, verde y azul en cada pixel (RGB). Estos datos son transformados a luma en un canal, $Y^{'}$, y crominancia en dos: C_B y C_R. Estos últimos representan la proporción de verde con respecto al azul y rojo, respectivamente. Este cambio es también útil dado que la luma se concentra en sólo un canal (en RGB está disperso entre los tres). Dado que la luma es el componente más importante para el ojo humano, se puede hacer una mayor compresión de datos.

Las componentes $Y^{'}C_BC_R$ son una combinación lineal de las componentes RGB. Las siguientes ecuaciones muestran más en profundidad la naturaleza de la conversión.

$$\begin{pmatrix} Y^{'} \\ C_B \\ C_R \end{pmatrix} = \begin{pmatrix} +0{,}2990 & +0{,}5870 & +0{,}1140 \\ -0{,}1687 & -0{,}3313 & +0{,}5000 \\ +0{,}5000 & -0{,}4187 & -0{,}0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} +1{,}00000 & +0{,}00000 & +1{,}40200 \\ +1{,}00000 & -0{,}34414 & -0{,}71414 \\ +1{,}00000 & +1{,}77200 & +0{,}00000 \end{pmatrix} \begin{pmatrix} Y^{'} \\ C_B - 128 \\ C_R - 128 \end{pmatrix}$$

31 de Mayo, 2012.

B. Separación en bloques

Luego de la transformación a los canales $Y^{'}C_BC_R$, la imágen se debe separar en bloques, que representan las "pequeñas zonas.[en] las cuales se desea eliminar las frecuencias altas de luma y croma. El tamaño de los bloques depende de la tasa de submuestreo de croma usada, y varía entre cuadrados de 8 pixeles y 16 pixeles. Dado que no se utilizó submuestreo, se utilizaron bloques de 8.

En el caso de que las dimensiones de la imágen no sean múltiplos del tamaño del bloque, se procede a extenderla hasta que los bloques quepan perfectamente. Los nuevos pixeles son llenados utilizando información de sus vecinos más inmediatos. Otras alternativas analizadas incluyen llenar los canales de los nuevos pixeles con 0, sin embargo se observaron mayores distorsiones en los bordes.

C. Transformación del Coseno Discreta

El siguiente paso consiste en transformar los valores de los canales en cada bloque a sus frecuencias espaciales en dos dimensiones, utilizando la transformación del coseno discreta. Esta última consiste en una transformada de Fourier discreta utilizando únicamente cosenos. Para una cierta matriz cuadrada, la transformada está dada por la ecuación:

$$G_{u,v} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{\alpha(u)\alpha(v)}{N} g_{x,y} \cos(\frac{\pi}{8}(x+\frac{1}{2})u) cos(\frac{\pi}{8}(y+\frac{1}{2})v) \tag{1}$$

Donde $g_{x,y}$ representa el elemento de la fila x y la columna y del bloque de entrada, y $G_{u,v}$ el elemento de la fila u y columna v de la matriz de frecuencias de salida.

Análogamente la antitransformada está dada por:

$$g_{x,y} = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \frac{\alpha(u)\alpha(v)}{N} G_{u,v} \cos(\frac{\pi}{8}(x+\frac{1}{2})u) cos(\frac{\pi}{8}(y+\frac{1}{2})v) \tag{2}$$

En ambas ecuaciones:

$$\alpha(k) = \begin{cases} \sqrt{1} & \text{si } k = 0 \\ \sqrt{2} & \text{en otro caso} \end{cases} \tag{3}$$

Nótese que los coeficientes de las frecuencias más bajas se encuentran en los valores bajos de x e y, es decir en la zona superior izquierda de la matriz de frecuencias. Similarmente los coeficientes de las frecuencias más altas se encuentran en la zona inferior derecha. Esta transformación concentra la señal de entrada en las frecuencias espaciales más bajas, es decir en la zona superior izquierda. En particular, comúnmente el mayor valor en módulo es $G_{0,0}$. Este valor representa la frecuencia con valor 0, y por lo tanto es un promedio de los 64 valores de entrada. A este valor se lo llama $DCcoefficient$ dado que en circuitos el promedio de una señal (es decir el valor asociado a la frecuencia cero) es la corriente efectiva directa (DC). Al resto de los valores se los llama AC debido a que representan la componente alterna de la corriente.

$$G = \begin{pmatrix} 54{,}245 & -117 & -7{,}7 & -12{,}81 & -0{,}28 & 0{,}06 & 0{,}128 \\ 26{,}113 & 6{,}13 & 0{,}73 & -0{,}1 & -5{,}13 & -0{,}9 & 0{,}1954 \\ -0{,}48 & 1{,}95 & -1{,}55 & -2{,}7 & -0{,}38 & -0{,}02 & 0{,}97 \\ 3{,}969 & 2{,}86 & 0{,}37 & -0{,}47 & 0{,}385 & 0{,}01 & 0{,}027 \\ 0{,}19 & 0{,}09 & 0{,}40 & 0{,}165 & -0{,}36 & 0{,}47 & 0{,}3 \\ -0{,}005 & -0{,}535 & -0{,}3 & -0{,}28 & 0{,}426 & -0{,}7 & 0{,}045 \\ 0{,}41 & 0{,}59 & 0{,}125 & -0{,}38 & -0{,}16 & -0{,}157 & 0{,}33 \\ 0{,}03 & -0{,}41 & -0{,}2 & 0{,}22 & 0{,}34 & 0{,}41 & 0{,}19 \end{pmatrix}$$

Ejemplo de una matriz salida, correspondiente a la luma del primer bloque de la imagen *Sample 1*. Nótese que los números de mayor magnitud se encuentran en la zona superior izquierda.

También es importante destacar que los valores de los canales son llevados a un rango alrededor del 0 antes de ser transformados, en el intervalo $[-127, 128]$. Esto se logra fácilmente restando 127 a todos los valores de entrada.

D. Cuantización

El paso final de la primer etapa de compresión, y el mayor responsable de la pérdida de información, es la cuantización de la matriz de coeficientes de frecuencias. Este paso consiste en dividir cada elemento de dicha matriz por un elemento de una matriz de cuantización, y redondear el valor obtenido. Las matrices de cuantización sugeridas por la especificación JPEG son las siguientes:

$$Qy = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

$$Qc = \begin{pmatrix} 17 & 18 & 24 & 47 & 47 & 99 & 99 & 99 \\ 18 & 21 & 26 & 26 & 66 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

La fórmula para obtener los valores es la siguiente:

$$\mathbf{B}(k,l) = round\left(\frac{\mathbf{A}(k,l)}{\mathbf{Q}_Y(k,l)}\right) \tag{4}$$

En otras palabras, se divide cada elemento de la matriz de coeficientes de frecuencas por su elemento correspondiente de la matriz de cuantización y se redondea. Para la luma se utiliza $\mathbf{Q}_Y(k,l)$ y para croma $\mathbf{Q}_c(k,l)$.

Nótese que existen varios valores de entrada que son transformados a la misma salida, es decir la relación no es inyectiva. Debido a esto, se pierde información. En particular, dado un coeficiente k de la matriz de cuantización, existe un intervalo de longitud k que producirá la misma salida. Ergo, valores más grandes de la matriz de cuantización implican una mayor pérdida de información. Si se observan ambas matrices con detenimiento, se puede

apreciar que los valores son mayores en la zona inferior derecha. Esto implica que la mayor pérdida de información ocurre en las frecuencias altas de los canales, que es lo que se buscaba en un principio. Asimismo, debido a que la transformada coseno discerta concentra la señal en la zona superior izquierda, y los coeficientes de cuantización son mayores en la zona inferior derecha, el resultado de la cuantización tendrá una densidad de ceros considerable en esta última zona.

$$By = \begin{pmatrix} 3 & -11 & -1 & -1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Ejemplo de una matriz tras ser cuantificada, correspondiente al primer bloque de la luma de *Sample 1*.

Se decidió además experimentar con distintas matrices de cuantificación. Para ello se multiplicó a las matrices estándares por un cierto valor α.

III. Codificación entrópica

A. Reordenamiento y codificación

La segunda etapa de la compresión consiste en aplicar una codificación entrópica a los datos cuantizados. Debido a la gran densidad de ceros producida por la transformación y cuantización, este tipo de algoritmos logra una compresión más efectiva. La especificación JPEG admite dos algoritmos: la codificación de *Huffman* y codificación aritmética. Ambos consisten en sustituir instancias de patrones frecuentes por cadenas de bits más cortas, con la diferencia que *Huffman* reemplaza cada patrón de entrada por su código respectivo, mientras que la aritmética lo transforma en un numero real. La codificación aritmética por lo general obtiene resultados entre 5 y 7 % más compactos, pero por lo general se utiliza *Huffman* debido a problema de patentes y su mayor velocidad.

Sin importar qué algoritmo se utilice, la gran densidad de ceros se traduce a que existe un patrón extremadamente frecuente, el cual puede ser reemplazado por una cadena de bits particularmente corta para reducir el tamaño del archivo. Sin embargo, se pueden modificar algunos datos para reducir aún más la tasa de compresión.

A.1 Reordenamiento

Antes de ejecutar el algoritmo de codificación, se pueden reordenar los valores de la matriz cuantificada, ordenando los valores según qué frecuencia representan.

Debido a que los valores asociados a mayores frecuencias son mayormente cero, este reordenamiento casuará cadenas largas de ceros en los datos antes de codificar. Esto resulta en una codificación más compacta. Además, por lo general se codifica un símbolo especial indicando que el resto del bloque está únicamente conformado por ceros.

A.2 Resta del coeficiente DC

El valor DC suele ser el mayor valor de cada bloque. Si se pueden acotar los valores de los bloques a un intervalo pequeño en un entorno del 0, la codificación sería más eficaz. Para eliminar los valores altos de DC, se puede hacer uso de que el DC representa el promedio de los canales de un bloque. Asumiendo una imagen normal, es muy posible que el promedio de los canales en pixeles adyacentes sea similar. Debido a esto, en vez de guardar el DC se puede guardar la diferencia entre el DC actual y el del pixel anterior. Esto reduciría la cantidad de valores posibles (pudiendo incrementar incluso la frecuencia de ceros marginalmente).

B. Cálculo de cota superior razonable

Para hallar una cota superior razonable y simple al tamaño del archivo resultante luego de la compresión, se puede utilizar la frecuencia relativa de valores de canales de pixeles con valor en 0. Utilizando una codificación de *Huffman*, se le podría asignar a estos pixeles un único bit, en 0. En el peor de los casos, el resto de los valores varía uniformemente entre -127 y 128, por lo cual se necesitan 9 bits para almacenarlos (un 1 seguido de la representación binaria del número).

En la imagen original, se utilizan 24 bits por pixel, 8 por cada canal. Asumiendo que p es la frecuencia de valores de canales en 0, es fácil verificar que en promedio el valor de un canal necesitará:

$$b_{pixel} = 3(p + 9(1 - p)) \tag{5}$$

Es fácil ver también que para valores de p mayores a $\frac{1}{8}$ ya se usan menos de 24 bits (nótese sin embargo que para valores tan bajos de p seguramente haya una mejor codificación que la asumida; esta última asume valores más cercanos a 1 de p, como se observa en la práctica). Con un valor de 0,9, por ejemplo, esta cantidad se reduce a $5,4$ bits.

Nótese que esta es simplemente una cota superior razonable. Utilizando restas de coeficientes DC y reordenamientos la cantidad de bits seguramente será menor. La estimación no contempla el espacio requerido por la tabla de codificaciones que se guarda en el encabezado del archivo. Este se puede despreciar asumiendo que la imagen posee dimensiones considerables.

IV. Procedimiento de recuperación

En esencia el procedimiento de recuperación consiste en los mismos pasos ya nombrados pero aplicados en orden inverso. Primero se descomprime utilizando la descodificación entrópica. Se multiplican los valores de cada bloque por su valor correspondiente en la matriz de cuantificación. Se aplica la antitransformada discreta del coseno a cada bloque, y luego se recupera la imagen en los canales luma y croma con los bloques. Finalmente se transforman dichos canales a los canales RGB.

V. Resultados y conclusiones

A. Tipos de imágenes

Como se puede observar en la Tabla 1, la codificación alcanza valores cercanos o superiores al 90 % de ceros en la imagen. Según la estimación planteada, esto reduce la cantidad de bits por pixel a valores entre 4 y 6. Asimismo, se puede observar que el error es más de tres veces mayor en las imágenes de tipo íconos, es decir alguna forma o figura sobre un fondo blanco, como el logo de *Facebook* o *Chrome*. Esto se debe a que en la frontera entre dicha figura y el fondo existen cambios brucos de luminosidad, que serán suavizados en la codificación. Esto incrementa el error cuadrático medio y hace que el contorno de la figura se distorsione. El error en el logo de *Chrome* es aún mayor, debido a ser circular. Dado que el algoritmo codifica en ventanas cuadradas, se puede esperar aún más distorsiones en el límite de la figura. En imágenes que representan fotografías de paisajes estos fenómenos no se encuentran habitualmente, por lo que se puede verificar que el error cuadrático medio es menor.

Imagen	**Tamaño en pixeles**	**Densidad de 0**	**Bits por pixel estimados**	**Error cuadrático medio**
Lenna	512x512	0.943	4.37	8
Sample1	74x56	0.91	5.4	11
Sample2	73x48	0.924	4.824	10
chrome	48x48	0.95	0.81	86
facebook	42x43	0.91	5.16	30

Tabla 1: Compresión y errores de imágenes tras aplicar la codificación, usando $\alpha = 1$.

B. Variaciones en la cuantización

En la tabla 2 se puede verificar que si se utilizan matrices de cuantificación con valores mayores (es decir mayor α) el error sube, pero la densidad de ceros lo hace también. Por otro lado para valores menores de α, tanto el error como la densidad de ceros disminuye.

α	**Densidad de 0**	**Bits por pixel estimados**	**Error**
0.5	0.74	9.24	
0.75	0.78	8.28	
1	0.81	7.56	
1.3	0.835	6.96	
1.5	0.845	6.72	
2	0.87	6.12	
3	0.894	5.27	

Tabla 2: Compresión y errores de imágenes tras aplicar la codificación, usando $\alpha = 1$.

Tabla 2: Uso de distinto α para la codificación de la imagen *chrome*.

C. Dimensiones indivisibles por 8

Para llevar una imagen a bloques cuyas dimensiones no son múltiplos de 8, se deben agregar pixeles. La tabla 3 muestra la diferencia entre el rellenado de dichos pixeles con ceros y el rellenado usando el pixel del borde más cercano.

Nombre	**Rellenado**	**Error cuadrático medio**
dog	Bordes	23
dog	Ceros	26
facebook	Bordes 17	
facebook	Ceros	30

Tabla 3: Comparación para distintos rellenados de imágenes, $\alpha = 1$.

Se puede verificar que si se rellena con ceros, el borde de la imagen presenta distorsiones de los colores verdes y magenta (ver imágenes 1 y 2).

Figura 2: Imagen 1. Rellenado usando pixel del borde. Imagen ampliada, factor 5. Factor $\alpha = 1$.

figure

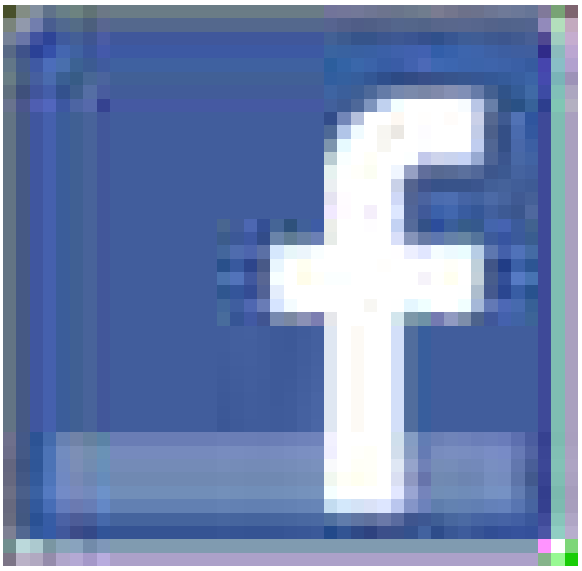

Figura 3: Imagen 2. Rellenado usando ceros. Imagen ampliada, factor 5. Factor $\alpha = 1$.
figure

D. Bloques en ampliaciones de imagenes

Si se amplía la imagen lo suficiente, se puede ver la división de bloques ocurrida (ver imagen 3 o Anexo A).

Figura 4: Imagen 3. Logo de *chrome* ampliado con un factor de 5. Factor $\alpha = 3$ para hacer más vistosos los bloques.
figure

Referencias

[1] *JPEG - http://en.wikipedia.org/wiki/Jpg*
[2] *Codificación aritmética - http://en.wikipedia.org/wiki/Arithmetic$_c$odingCodificaciónHuffman – http : //en.wikipedia.org/wiki/Huffman$_c$oding*
[3] Codificación Huffman - http://en.wikipedia.org/wiki/DC$_B$iasJPEG – *http : //www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/jpeg/jpeg/en*

Anexo A: Imagenes y sus codificaciones. Factor $\alpha = 1$ en todos los casos.

Figura 5: Imagen de Lenna de 512x512 pixeles

figure

Figura 6: Imagen de Lenna de 512x512 pixeles tras codificar

figure

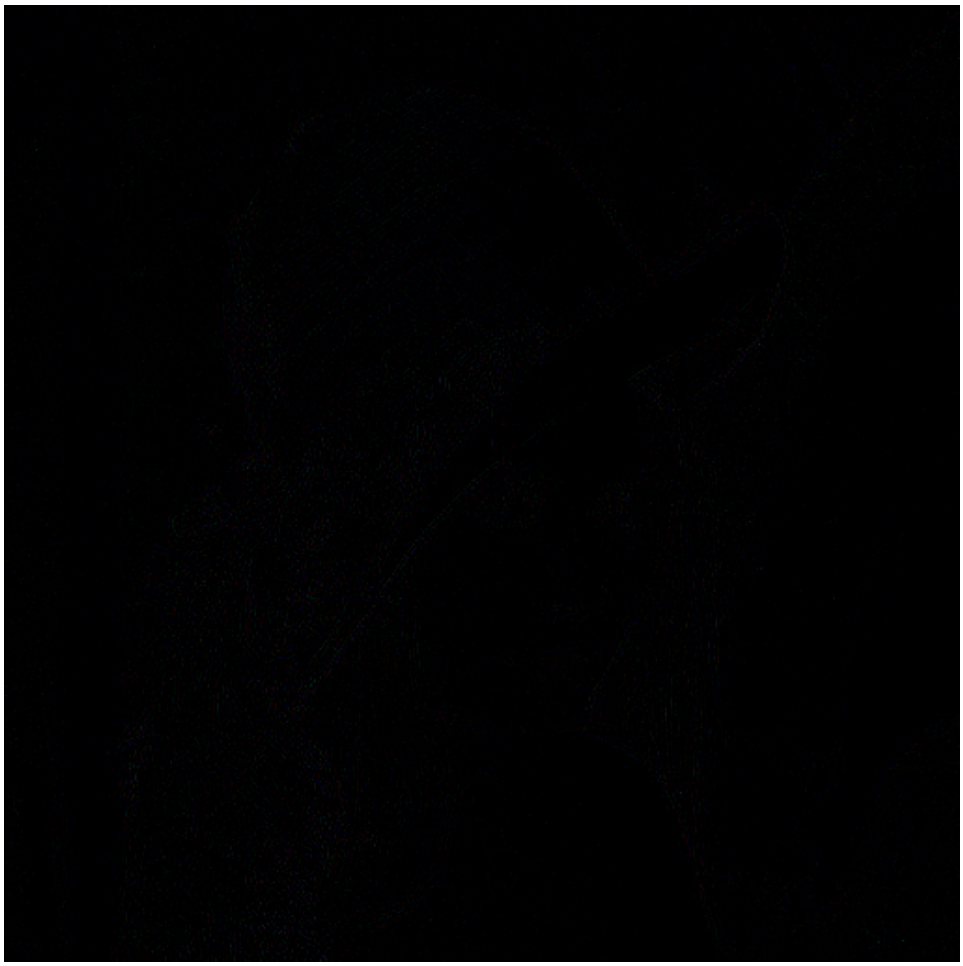

Figura 7: Imagen de Lenna de 512x512 pixeles

figure

Figura 8: *Sample 1*, ampliada factor 5.

figure

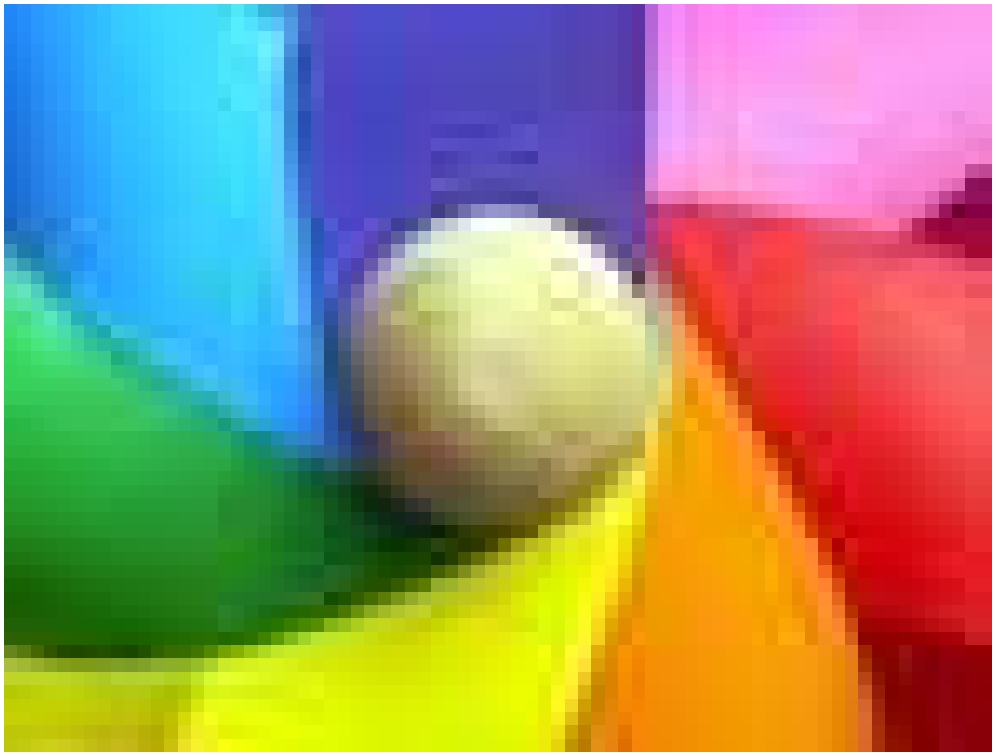

Figura 9: *Sample 1* tras la codificación, ampliada factor 5.

figure

Figura 10: *Sample 3*, ampliada factor 5.

figure

Figura 11: *Sample 3* tras la codificación, ampliada factor 5. Se pueden observar ciertos bloques en el centro de la imagen.

figure

Anexo B: Variación del factor *alpha*.



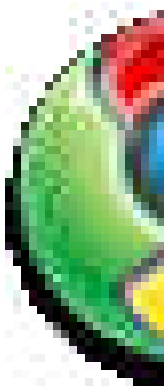



Figura 12: Codificación de varias imágenes del logo de *Chrome* utilizando distintos valores de *alpha*
figure

Anexo C: Código Octave

E. Pasaje de espacio de color de una imagen de RGB a $Y'C_bC_r$

```
function ret = rgb_to_ybr(M)
        coef_matrix = [ 0.2990, 0.5870, 0.1440; -0.1687, -0.3313, 0.5; 0.5, -0.4187, -0.0813];
        sum_matrix = [0;128;128];
        rgb = zeros(3,1);
        ret = zeros(size(M));
        for i = 1:size(M,1)
                for j = 1:size(M,2)
                        for k = 1:3
                                rgb(k) = M(i,j,k);
                        end
                        aux = coef_matrix * rgb + sum_matrix;
                        for k = 1:3
                                ret(i,j,k) = aux(k);
                        end
                end
        end
endfunction
```

F. Pasaje de espacio $Y'C_bC_r$ a RGB.

```
function ret = ybr_to_rgb(M)
        coef_matrix = [ 1, 0, 1.4020; 1, -0.34414, -0.71414; 1, 1.772, 0];
        ybr = zeros(3,1);
        ret = zeros(size(M));
        for i = 1:size(M,1)
                for j = 1:size(M,2)
                        ybr(1) = M(i,j,1);
                        ybr(2) = M(i,j,2) - 128;
                        ybr(3) = M(i,j,3) - 128;
                        aux = coef_matrix * ybr;
                        for k = 1:3
                                ret(i,j,k) = round(aux(k));
                        end
                end
        end
endfunction
```

G. Pasaje de imagen a bloques $8x8$.

```
function ret = img_to_blocks(M,n)
        width = size(M,2);
        height = size(M,1);
        s = ceil(width/n);
        width2 = s*n;
        height2 = ceil(height/n) * n;
        blocks = s * ceil(height/n);
        ret = zeros(blocks, 3, n,n);
        for i = 1:height2
                h = floor((i-1)/n);
                i2 = round(i-h*n);
                for j = 1:width2
                        w = floor((j-1)/n);
                        block = round(w + h*s)+1;
                        j2 = round(j-w*n);
                        for k = 1:3
                                if( width >= j && height >= i)
                                        ret(block, k, i2, j2) = M(i,j,k);
```

```
                                else
                                        if( j > height && i > height)
                                                ret(block, k, i2, j2) = M(height,width,k);
                                        else
                                                if( j > width)
                                                        ret(block, k, i2, j2) = M(height,width,k);
                                                endif
                                                if( i > height)
                                                        ret(block, k, i2, j2) = M(height,j,k);
                                                endif
                                        endif
                                endif
                        end
                end
        end
endfunction
```

H. Pasaje bloques a imagen.

```
function ret = blocks_to_img(M, height, width)
        n = size(M,3);
        blocks = size(M,1);
        s = ceil(width/n);
        ret = zeros(height,width, 3);
        for i = 1:blocks
                aux = floor((i-1)/s);
                j2 = round(aux * n);
                k2 = round(((i-1) - aux*s)*n);
                for j = 1:n
                        j3 = j + j2;
                        for k = 1:n
                                k3 = k +k2;
                                if( j3 <= height && k3 <=width)
                                        for l = 1:3
                                                ret(j3,k3,l) = M(i,l,j,k);
                                        end
                                endif
                        end
                end
        end

endfunction
```

I. Transformada coseno discreta de bloque.

```
function ret = disc_cos_transf(M, N)
        ret = zeros(N,N);
        for l = 1:N
                if(l == 1)
                        a = 1;
                else
                        a = sqrt(2);
                endif
                for k = 1:N
                        if(k == 1)
                                b = 1;
                        else
                                b = sqrt(2);
                        endif
                        sum = 0;
```

```
                        for n = 1:N
                                for m = 1:N
                                        sum = sum + a*b*M(n,m)*cos(pi*((n-1/2)*(l-1))/N)*cos(pi*((m-1/2)*(k-
                                end
                        end
                        sum = sum/N;
                        ret(l,k) = sum;
                end
        end
endfunction
```

J. Antitransformada coseno discreta de bloque.

```
function ret = disc_cos_antitransf(M, N)
        ret = zeros(N,N);
        for n = 1:N
                for m = 1:N
                        sum = 0;
                        for l = 1:N
                                if(l == 1)
                                        a = 1;
                                else
                                        a = sqrt(2);
                                endif
                                for k = 1:N
                                        if(k == 1)
                                                b = 1;
                                        else
                                                b = sqrt(2);
                                        endif
                                        sum = sum + a*b*M(l,k)*cos(pi*((n-1/2)*(l-1))/N)*cos(pi*((m-1/2)*(k-
                                end
                        end
                        sum = sum/N;
                        ret(n,m) = sum;
                end
        end
endfunction
```

K. Cálculo del error cuadrático medio de una diferencia de imágenes.

```
function ret = calc_error(M)
        M = uint32(M);
        height = size(M,1);
        width = size(M,2);
        ret = 0;
        for i = 1:height
                for j = 1:width
                        for k = 1:3
                                ret = ret + (M(i,j,k) * M(i,j,k));
                        end
                end
        end
        ret = ret / (height * width * 3);
endfunction
```

L. Cálculo de la densidad de ceros.

```
function ret = calc_zeros(B)
        blocks = size(B,1);
        ret = 0;
```

```
        N = size(B,3);
        for b = 1:blocks
                for k = 1:3
                        for i = 1:N
                                for j = 1:N
                                        if(B(b,k,i,j) == 0)
                                                ret = ret+1;
                                        endif
                                end
                        end
                end
        end
        ret = ret / (blocks * N * N * 3);
endfunction
```

M. Antitransformada coseno discreta de bloque.

```
function ret = disc_cos_antitransf(M, N)
        ret = zeros(N,N);
        for n = 1:N
                for m = 1:N
                        sum = 0;
                        for l = 1:N
                                if(l == 1)
                                        a = 1;
                                else
                                        a = sqrt(2);
                                endif
                                for k = 1:N
                                        if(k == 1)
                                                b = 1;
                                        else
                                                b = sqrt(2);
                                        endif
                                        sum = sum + a*b*M(l,k)*cos(pi*((n-1/2)*(l-1))/N)*cos(pi*((m-1/2)*(k-
                                end
                        end
                        sum = sum/N;
                        ret(n,m) = sum;
                end
        end
endfunction
```

N. Codificación y decodificación de una imagen.

```
function encode_and_decode(inimg, outimg, diffimg, alpha)
        %First we encode...
        I = imread(inimg);
        B = rgb_to_ybr(I);
        B = img_to_blocks(B,8);
        aux = zeros(8,8);
        for b = 1:size(B,1)
                for k = 1:3

                        for i = 1:8
                                for j = 1:8
                                        aux(i,j) = B(b,k,i,j) - 128;
                                end
                        end
                        aux = disc_cos_transf(aux, 8);
```

```
                        for i = 1:8
                                for j = 1:8
                                        B(b,k,i,j) = aux(i,j);
                                end
                        end
                end
        end
        B = quantify(B,alpha);
        %Done encoding! Calculate stuff here...
        zeros = calc_zeros(B)
        %Now decode
        B = dequantify(B,alpha);
        for b = 1:size(B,1)
                for k = 1:3
                        for i = 1:8
                                for j = 1:8
                                        aux(i,j) = B(b,k,i,j);
                                end
                        end
                        aux = disc_cos_antitransf(aux, 8);
                        for i = 1:size(aux,1)
                                for j = 1:size(aux,2)
                                        B(b,k,i,j) = round(aux(i,j) + 128);
                                end
                        end
                end
        end
        %zeros = calc_zeros(B)
        B = blocks_to_img(B, size(I,1), size(I,2));
        B = ybr_to_rgb(B);
        B = uint8(B);
        mean_square_error = calc_error(I-B)
        imwrite(B,outimg);
        imwrite(abs(I-B), diffimg);
endfunction
```