Comprimidor de Imágenes Utilizando Estándar JPEG 2000

Joaquín Farias, *Estudiante*

Resumen—En Este informe se presenta una aplicación para comprimir imágenes utilizando el estándar JPEG 2000. Se explicará la descomposición y la reconstrucción de la imagen ocupando este método. Se especificará la cuantización ocupada y se hará un análisis del error de reconstrucción con respecto al porcentaje de compresión.

Palabras clave—Comprimidor de imágenes, JPEG 2000, Wavelets, Procesamiento de imágenes.

1 Introducción

Un grupo de expertos llamados ***Joint Phografic Experts Group*** fue el que creo el formato más famoso, hoy por hoy, para compresión de imágenes, este formato lleva sus siglas **JPEG**, basado en transformada discreta del coseno. Luego, en el año 2000 intentaron mejorar el formato antes creado, esto dio origen al formato **JPEG 2000**, que esta vez está basado en la compresión mediante *Wavelets* o bancos de filtros.

El formato **JPEG 2000** tiene una clara ganancia de compresión respecto al anterior formato JPEG. Algunas de las ganancias son, por ejemplo, que ofrece una compresión tanto con pérdida y sin pérdida en el flujo mismo archivo, mientras que JPEG utiliza en general solamente con pérdida. Otra ventaja es su capacidad de mostrar imágenes a diferentes resoluciones y tamaños, desde el mismo archivo de imagen. Con JPEG, un archivo de imagen sólo es capaz de ser desplegado de forma individual, con una resolución determinada. Debido a que JPEG 2000 se basa en wavelets, la corriente de wavelet puede ser sólo parcialmente descomprimida si el usuario quiere una imagen de baja resolución, mientras que también puede mostrar la imagen de resolución completa si se desea.

2 Motivación y descripción del algoritmo de formato JPEG

El formato JPEG 2000 cuenta con varios pasos estándar para la compresión de la imagen.

El primero de estos pasos, aunque opcional, es el traspasar la imagen desde el formato RGB (o en el que se encuentre) al formato YCbCr, luego de forma opcionalmente se pueden encoger las matrices de crominancia para aprovechar la cualidad del ojo humano de notar más las diferencias de brillo que de color, así se puede comprimir más aun la imagen perdiendo información que el ojo humano no notará.

El segundo de estos pasos es utilizar la transformada de *Wavelets* de forma reiterativa para comprimir la imagen. Estos filtros pueden ser de distinto tipo. Existen varias familiar creadas para la compresión, pero no se queda exento de poder crear un filtro propio que se acomode al uso que se desea.

El tercero de estos pasos es cuantizar la imagen mediante alguna fórmula creada para un caso específico o general. Al igual que en el caso de los filtros, existen varios tipos de cuantizaciones, pero se puede crear una específica para cierto uso deseado.

El cuarto paso es codificar la matriz recorriéndola en zigzag para luego aplicarle la codificación de Huffman. Esto debido a que, según estudios hechos por los creadores del formato, el recorrer la matriz de la imagen comprimida en zigzag tiene una mayor probabilidad de tener elementos iguales contiguos.

2.1 Formato YCbCr

El formato de imagen YCbCr se compone de tres matrices: una que guarda la información de luma (Y) y dos que guardan la información de crominancia (Cb y Cr).

Como opción se pueden encoger las matrices de crominancia y aprovechar la cualidad del ojo humano de notar más la diferencia de brillo que de color, así existen formatos que encogen la matriz a la mitad de su valor, así como otro que lo hace una a cuarto de su valor y la otra a la mitad.

En el caso de nuestro algoritmo **encogimos las dos matrices de crominancia a la mitad de sus dimensiones**.

3 Transformada de Wavelets

Luego de tener la imagen en formato YCbCr se hace la **transformada de Wavelets,** como se dijo anteriormente esta transformada ocupa filtros de forma reiterativa para reordenar la imagen y acercar las altas frecuencias hacia el cero utilizando un filtro pasa altos. También, se utilizan tres distintos filtros pasa bajos para tener finalmente la imagen divida en cuatro imágenes más pequeñas que representan de distinta manera a la imagen original. Luego, se puede reiterar el paso anterior ocupando los filtros en la imagen pequeña en que se utilizó el filtro pasa bajos, esto se ilustra en la Fig. 1. Estas reiteraciones pueden hacer las veces que se requiera según el caso.

Todo el procedimiento antes mencionado se ejecuta en la función **WAVELETS** dentro de nuestro programa que ocupa la función dada por *Matlab* llamada *wavedec2*.

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

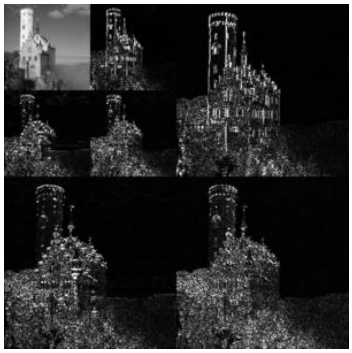


Fig. 1. Ejemplo de transformada de Wavelets con dos reiteraciones en una imagen de un solo canal de escala de grises.

4 CUANTIZACIÓN Y MATRIZ DE CUANTIZACIÓN

Para cuantizar nuestra imagen con la transformada de *wavelets* ya aplicada es necesario idear alguna forma de fija o dinámica para hacerlo. En el caso del algoritmo acá planteado se hace mediante un factor fijo de cuantización que varía dependiendo de a cuáles canales se está cuantificando.

Así, la función que cuantiza al canal Y, de luminancia, seria Q. Este factor dividía a toda la matriz Y para luego redondear los resultados hacia abajo, así

$$Y_{cuantizada} = Q \cdot \left[\frac{Y}{Q}\right]$$

Para los canales de crominancia se cambia esta cuantización, ya que puede ser un poco más agresiva sin afectar mucho la calidad de imagen, Después de hacer algunas pruebas se elige tener una cuantización que también dependa del valor de Q, pero sea un poco más agresivo a medida que Q aumente. Se eligió de aumentar de forma logarítmica, así la función que cuantiza a los canales de crominancia Cb y Cr, seria

$$Cb_{cuantizada} = \log(Q) \cdot Q \cdot \left[\frac{Cb}{\log(Q) \cdot Q}\right]$$

$$Cr_{cuantizada} = \log(Q) \cdot Q \cdot \left[\frac{Cr}{\log(Q) \cdot Q}\right]$$

Luego, este factor S aplica la siguiente formula a la matriz de cuantización MQ

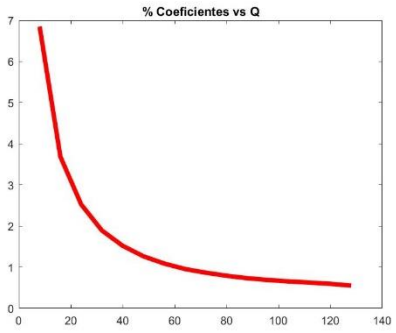
$$MQ' = \left[\frac{S \cdot MQ + 50}{100}\right]$$

Todo lo anterior se hace en la función **CUANTIZAR** del programa.

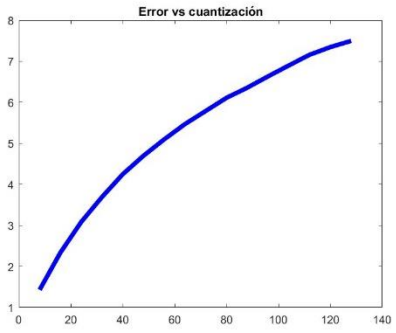
Como toda esta cuantización depende del factor Q, se hizo un estudio a cómo cambia el error y el porcentaje de coeficientes no nulos a medida que cambia este factor. Este estudio para una imagen RGB traspasada YCbCr se muestra en la Fig. 2.

Este estudio nos sirve para tener una idea de cuanta seria la compresión ideal para afectar lo menos posible a la fidelidad de la imagen.

Pero, se necesita crear un parámetro que relaciones el



(a)



(b)

Fig. 2. Graficas sobre estudio de cantidad de cuantización. (a) Grafica del factor de cuantización Q versus el porcentaje de coeficiente no nulos. (b) Grafica del factor de cuantización Q versus el porcentaje de error.

factor Q con el porcentaje de compresión. Así se llega a la gráfica mostrada en la Fig. 3, donde se llega a la siguiente ecuación del factor Q que depende de porcentaje de compresión C

$$Q = 5x10^{-13}e^{0,3836 \cdot C}$$

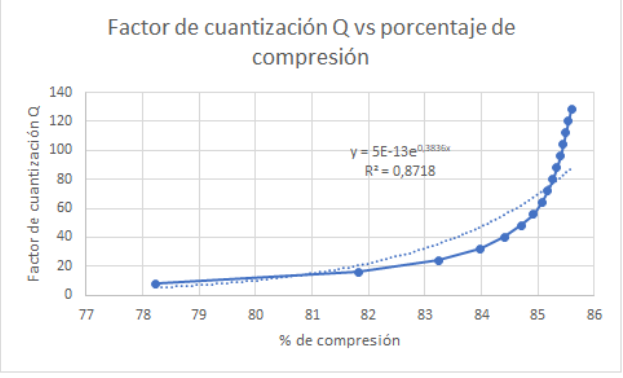


Fig. 2. Gráfico entre factor de calidad Q versus el porcentaje de compresión.

5 ALGORITMO DE CODIFICACIÓN

Para la codificación se recorrió toda la matriz en **zigzag** para luego ocupar el **algoritmo de Huffman [4]**, el cual genera un diccionario binario para cada símbolo que se requiera. Luego de formar un diccionario para cada canal se convierten todos estos, notando que el diccionario cambia para cada canal. Finalmente, se guarda todo una un solo vector, además de las dimensiones de cada de cada canal.

Se guardan estos datos codificados en binario en un archivo .bin con la función **fwrite** de *Matlab,* además se guardan los encabezados necesarios para la decodificación en un archivo .dat con la función **save** de *Matlab*.

Todo lo anterior está asociado a la función CODIFICAR de nuestro código.

6 ALGORITMO DE DECODIFICACIÓN Y RECONSTRUCCIÓN

Para hacer el procedimiento de decodificación y reconstrucción solo se necesita hacer le procedimiento inverso de las funciones **WAVELETS** y **DECODIFICAR**.

En el procedimiento inverso de decodificación se leen los datos del archivo .bin y los encabezados del archivo .dat para volver a formar las matrices de datos con su tamaño original, luego se les aplica la función inversa de la codificación de huffman con el diccionario guardado como encabezado. La función asociada a este procedimiento es **DECODIFICAR.**

En el procedimiento inverso de la función **WAVELETS** se realiza la transformad inversa de wavelets ocupando el mismo filtro y sabiendo la cantidad de veces que se itero en la transformada. Esto se hace gracias a una función que proporciona *Matlab* llamada *waverec*2. Finalmente se llevan las matrices de crominancia a su tamaño original, es decir las mimas dimensiones que la matriz. Este procedimiento se realiza en la función **IWAVELETS.**

7 PRUEBAS Y ANÁLISIS

Dentro de este capítulo se presentarán los resultados de la aplicación de nuestro programa y el análisis de error.

7.1 Resultados y ejemplos de reconstrucción de imagen comprimida

Los resultados de pruebas hechas con una imagen con distintos niveles de compresión se muestran en la Fig. 6. Se nota que el resultado es bastante bueno, pero a medida que más se comprime la imagen el error aumenta.

6 CONCLUSIONES

Los resultados obtenidos en este informe son satisfactorios, ya que se alcanzó un buen nivel de compresión sin comprometer tanto error en la imagen comprimida con respecto a la original.

También se notó que el tiempo computacional que tiene este algoritmo no es excesivamente grande, ya que solo funciona en base a iteración de funciones básica, aunque

(a)

(b)

(c)

Fig. 4 Ejemplo del algoritmo con una imagen. (a) Imagen original. (b) Imagen con Q = 85. (c) imagen con Q = 50.

depende considerablemente a que tan complejos sean los filtros que se ocupen para hacer la transformada de *Wavelets*.

Se estudio el error con distintos tipos de filtros y maneras de cuantificación, explorando cuales son buenos y malos dependiendo de la fidelidad de imagen que se desee.

Por último, es bueno resaltar el bueno funcionamiento del algoritmo de Huffman cuando los datos están ordenados de manera de optimizarlo.

REFERENCIAS

[1] Boyd, S., & Vandenberghe, L. (2018). Introduction to applied linear algebra: vectors, matrices, and least squares. Cambridge university press.

[2] Norvig, P., & Russell, S. (2004). Inteligencia artificial. Editora Campus, 20.)

[3] Independent JPEG Group. (12 de junio de 2020), disponible en: https://www.ijg.org/.

[4] Knuth, D. E. (1985). Dynamic huffman coding. Journal of algorithms, 6(2), 163-180.

Joaquín Farias Muñoz Estudiante de ingeniería civil electrónica, Pontificia Universidad catolica de Valparaíso.