Comprimidor de Imágenes Utilizando Estándar JPEG

Joaquín Farias, *Estudiante*

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

Palabras clave—Comprimidor, JPEG, Imagen, 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**. Además, es considerado también de forma común como un formato de imágenes.

El formato JPEG es usado por la mayoría de las cámaras fotográficas actuales y permite definir el grado de compresión dependiendo de la aplicación. Es preferido por sobre otros formatos de compresión sin perdida como **PNG** porque está pensado para imágenes grandes y mucha variación cromática, pero no funciona tan bien imágenes con pocas variaciones cromáticas y espacios considerables de un solo color.

2 Motivación y descripción del algoritmo de formato JPEG

Este cuenta con varios pasos estándar para la compresión de la imagen.

El primero de estos pasos es la traspasar la imagen desde al formato YCbCr, luego de forma opcional 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 dividir las matrices de la imagen en matrices de 8x8 y aplicarles la transformada de coseno discreta (DCT).

El tercer de estos pasos es cuantizar la imagen mediante una matriz de cuantización Q. Este paso a pesar de que es estándar, las matrices Q no lo son y pueden variar dependiendo de la aplicación.

El cuarto paso es codificar la matriz recorriéndola en zigzag para luego aplicarle la codificación de Huffman.

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 del coseno (DCT)

Luego de tener la imagen en formato YCbCr se **divide la imagen en matrices pequeñas de 8x8** y se hace la transformada de coseno a los tres canales (o matrices). Para aplicar esta transformada se debe, primero definir una matriz DCT que se logra gracias a la función que entrega *MATLAB dctmtx*. También, en nuestro algoritmo restamos el valor medio de cada canal a todos los valores del canal para evitar valores muy dispares en la codificación.

Todo el procedimiento antes mencionado se ejecuta en la función **DCT**.

4 Cuantización y matriz de cuantización

Para cuantizar nuestra imagen con la DCT ya aplicada es necesario diseñar una matriz de cuantización MQ por la cual se dividirán todos nuestros bloques de 8x8. En este programa se ocupa la matriz de cuantización propuesta en [3], que además varía dependiendo de un factor de calidad Q. Así, se define la matriz

$$MQ = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 76 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Luego, el factor de calidad Q transforma la matriz aplicándole una operación matemática, pero el factor Q primero transforma otro factor S que depende de él, así

$$S = \begin{cases} \frac{5000}{Q} & Q < 50 \\ 200 - 2 \cdot Q & Q \geq 50 \end{cases}$$

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

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

Ahora con nuestra matriz de cuantización definida, dividimos cada bloque de 8x8 por esta matriz para cuantizar. En la Fig. 1 se muestra cómo cambia la compresión respecto al factor de calidad Q ingresado.

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

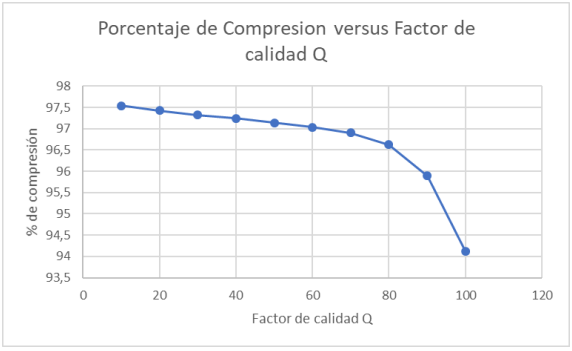

Fig. 1. Gráfico que muestra la relación entre el factor de calidad Q ingresado y la compresión de la imagen.

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 **DECUANTIZAR** y **CODIFICAR**.

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 decuantizar se multiplica los bloques por la matriz de cuantización MQ' para luego aplicarle la transformada de coseno inversa. Finalmente se llevan las matrices de crominancia a su tamaño original, es decir las mimas dimensiones que la matriz Y, para luego sumarle el valor medio de la matriz al principio restado. Este procedimiento se realiza en la función **DECUANTIZAR.**

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.

7.2 Análisis de error

Para analizar el error entre la imagen original y la comprimida se utilizó el error cuadrático entre todos los pixeles de la imagen. En la Fig. 3. Se compara el error con la

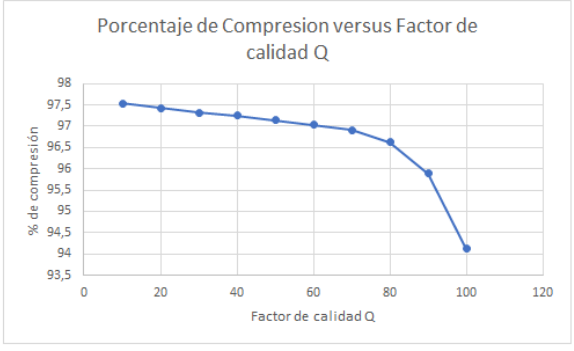

Fig. 2. Gráfico entre factor de calidad Q versus el porcentaje de compresión.

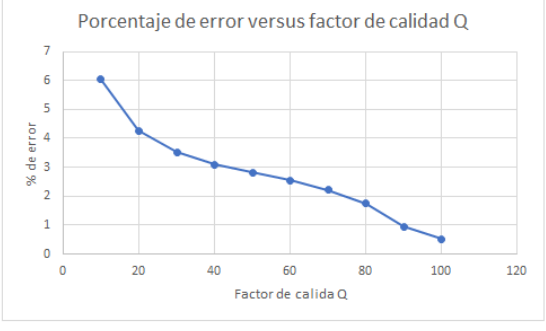

Fig. 3. Grafica del factor de calidad Q versus el porcentaje de error.

compresión y en la Fig. 4 se compara el error con el factor de calidad Q.

Se debe notar que el error es muy bajo y que a medida que el factor de calidad aumenta el error también lo hace. También, se nota que a medida que el factor de calidad aumenta menor es el porcentaje de compresión.

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 lleva tiene este algoritmo no es excesivamente grande, ya que solo funciona en base a iteración de funciones básicas.

Se noto que la gran diferencia entre una compresión buena y una mala es la matriz de cuantización que se ocupa. Además, esta matriz de cuantización puede variar para la matriz de luma y las de crominancia para obtener un valor aún mejor, caso que no se estudió dentro de este informe.

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.

(a)

(b)

(c)

Fig. 4 Ejemplo del algoritmo con una imagen. (a) Imagen original. (b) Imagen con Q = 60. (c) imagen con Q = 20.