Filtrado y compresión de video

Francisco González[1] and Sebastián Durán[1]

[1]Affiliation not available

May 4, 2019

Abstract

This document shows a simple way to filter, compress and encode a video focused on the transmission of images through a small bandwidth channel. It will be shown how we approach this problem by developing a small program in Python that manages to compress coding images to be sent and its inverse process in order to recover it.

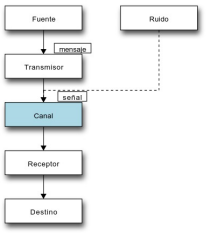

Figure 1: Modelo de comunicación de Shannon

Resumen

Este documento muestra una forma simple de filtrar, comprimir y codificar un vídeo enfocado en la transmisión de imágenes a través de un canal de ancho de banda pequeño. Se mostrará como abordamos este problema desarrollando un pequeño programa en Python que logre comprimir codificar imágenes para ser enviadas y su proceso inverso para poder recuperarla.

Introducción

Claude E. Shannon propuso un modelo para describir la comunicación. Este modelo muestra cómo la información (o mensaje) es generada y enviada por el emisor y recibida por el receptor.

La problemática a resolver consta de un emisor que recibe un video de una cámara IP. Este video llega al emisor con un ruido y debe ser transmitido a un receptor a través de un canal con ancho de banda acotado. Para lograr resolver esta problemática se ha decidido descomponer la solución en los siguientes pasos:

1. Filtrar la imagen original para eliminar potenciales ruidos
2. Transformar la información sobre la imagen del espacio de píxeles al espacio de frecuencia.
3. Cuantizar la información para aumentar la redundancia de datos pertenecientes a la información de la imagen (esto ayuda a reducir su tamaño).
4. Codificar la información para poder ser transmitida a través de un canal

Los pasos previos corresponden al trabajo realizado por la parte emisora del mensaje. Para el receptor, en cambio, es necesario realizar los procesos inversos para poder obtener imágenes a partir de los datos recibidos que están codificados (decodificar, decuantizar y destransformar para obtener la imagen resultante).

Metodología

Representación de imágenes digitales

Una imagen digital es representada por una matriz M de dimensión nxm donde cada elemento $M_{i,j}$ representa un píxel en colores (generalmente en formato RGB). Para este problema en específico se utilizará una imagen en escala de grises cada elemento de la matriz representará un nivel de luminiscencia donde 0 representa la oscuridad total y 255 la luz máxima.

JPEG

Para el desarrollo de nuestro trabajo nos hemos basado en los procesos de compresión y cuantización realizados por el estándar JPEG[1].

Filtrado de imagen

Una vez capturado un fotograma desde la cámara podemos ver la siguiente imagen:

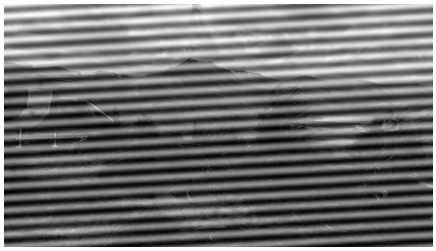

Figure 2: Imágen con ruido periódico

Al observar el notable ruido en la imagen capturada, es necesario revisar el espectro de magnitud de ella (utilizando la transformada rápida de Fourier discreta) para intentar encontrar algo que nos indique donde se ubica el ruido periódico que estamos buscando.

Una vez localizados los puntos (en el dominio de la frecuencia) que provocan el ruido periódico,

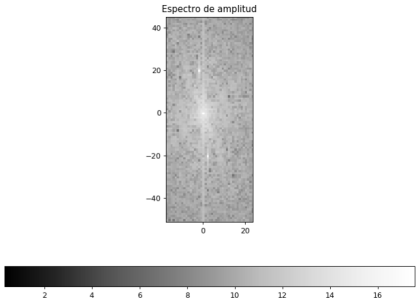

Figure 3: Ruido periódico en espectro de magnitud

logramos corregirlo promediando las zonas afectadas por esos puntos (en el dominio de frecuencia) con los valores de sus vecinos.

Figure 4: Imagen original filtrada

Transformación

Para realizar la transformación, hemos utilizado la transformada coseno discreta (DCT en adelante) que servirá para transformar la información desde el dominio de píxeles al dominio de frecuencias. La DCT tiene como ventaja principal el ser capaz de concentrar la mayor parte de la información en pocos coeficientes que han sido transformados[2]. Esto permite que la codificación resulte mas eficiente que al utilizar por ejemplo la transformada de Fourier.

La imagen es dividida en bloques de tamaño reducido sobre las cuales se aplica la DCT, en nuestro caso, la hemos dividido en bloques de tamaño 8x8 píxeles.

Cuantización

Es aquí donde ocurre la mayor parte del proceso de compresión, ya que parte de la información es descartada dejando sólo los coeficientes más representativos.

Para realizar la cuantización se divide cada bloque de 8x8 por una matriz Q que tiene su misma dimensión. La matriz Q es llamada matriz de cuantización[3], que consta de componentes numéricos cuyos valores representan la importancia que se le da a cada intervalo de frecuencia en el espectro de la imagen.

La operación de cuantización puede expresarse de la siguiente forma:

$$M_q\,(i,j) = redondeo(\tfrac{M(i,j)}{Q(i,j)})$$

donde M_q es la matriz que resulta del proceso de cuantización, M es la matriz resultante al aplicar DCT y Q es la matriz de cuantización. Cabe destacar que la división se realiza elemento a elemento (como el producto Hadamard). Cada valor resultante de la división es redondeado al entero más próximo.

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

Figure 5: Matriz de cuantización Q

Codificación

Para realizar el proceso de codificación, utilizamos la codificación de Huffman[4], la cual a partir de la generación de una tabla de códigos (diccionario) de largo variable permite codificar un símbolo en una secuencia binaria. Esto nos permite representar toda la información en una tira binaria. En el proceso de codificación también ocurre la compresión de datos.

Generación del diccionario y árbol de Huffman

Es necesario crear el árbol de Huffman para poder generar nuestro diccionario. El árbol de Huffman es un árbol binario en las que cada una de sus hojas es uno de los símbolos contenidos dentro de la matriz de datos cuantizados.

El algoritmo dice lo siguiente:

1. Se ordena cada símbolo según su probabilidad
2. Se escogen los dos símbolos con menor probabilidad y se agrupan, formando un nuevo nodo. Ese nodo representa la suma de las probabilidades de los dos elementos que fueron agrupados. El nuevo nodo se convertirá en el padre de los nodos escogidos, donde el lado izquierdo (representado por un 0) conectará al hijo con mayor probabilidad y el lado derecho (1) al de menor probabilidad.
3. Se repite el paso 2 hasta que el árbol se reduzca a dos conjuntos, los que serán conectados por un nuevo nodo padre de la misma forma que el paso 2.
4. Para obtener el código binario de cada hoja basta con realizar el recorrido desde la raíz del árbol concatenando los 0's y 1's en el orden que aparecen.

Una vez generado nuestro diccionario podemos recorrer nuestra matriz elemento por elemento (de izquierda a derecha y de arriba a abajo), cambiar el elemento por su valor binario y preparar nuestra tira binaria.

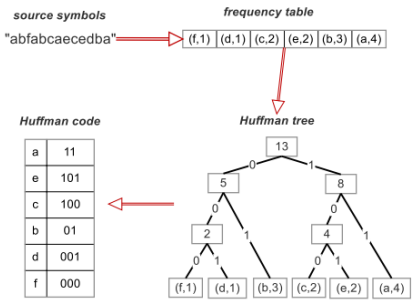

Figure 6: Generación del diccionario (Huffman code) a partir del árbol de Huffman

Decodificación

Para realizar la decodificación se almacena la cadena de bits recibida junto al diccionario y se procede a recorrer y tomar un extracto de esta cadena desde la primera posición con un largo inicial de 1. Si el extracto de bits no se encuentra en el diccionario, se agrega al extracto el siguiente bit en la cadena para volver a consultar al diccionario. En caso de que el extracto se encuentre, se obtiene el valor correspondiente y se almacena en una lista de valores. Se repite el proceso hasta que la tira de bits quede vacía. Una vez leída la tira de bits completa, se transforma a la matriz (conociendo previamente la cantidad de filas y columnas) donde cada m elementos forman una fila de la matriz que se quiere rearmar.

Decuantización

Una vez rearmada la matriz, es necesario decuantizarla, por lo que realizamos el proceso inverso de la cuantización realizada por la parte emisora del mensaje: Obtenemos submatrices de 8x8 y las multiplicamos término a término con la matriz Q (debe ser la misma matriz Q). Cada resultado es redondeado al entero más cercano.

Transformada discreta de coseno inversa

Ahora queda aplicar a cada submatriz la transformada de coseno inversa (IDCT en adelante) para obtener los valores referente a los pixeles de la imagen.

Aplicada la IDCT hemos obtenido la imagen recibida decodificada.

Resultados y análisis

Sin menospreciar el logro de codificar y decodificar una imagen, el tema de importancia es que tanto se pudo comprimir para poder ser enviada a través del canal de ancho da banda limitado.

Se procesó la totalidad de los cuadros del video para poder llegar a un valor promedio de bits correspondiente a los datos codificados.

Los resultados obtenidos en cuanto a la cantidad de bits de la imagen original y los datos codificados son los siguientes:

Sean

- W_{img}: "Peso" calculado de la imagen original filtrada(en bits)
- W_{data}: "Peso" calculado de la secuencia de bits de los datos codificados

donde:

- El peso de la imagen original filtrada está dado por la cantidad de pixeles de alto, ancho y número de bits por valor (8 bits):

$W_{img} = width * height * 8 = 3256320[bits]$

$W_{data} = largo(data) = 509282[bits]$

A partir de estos valores de bits, es posible obtener una relación entre el peso de la imagen original y los datos comprimidos:

$W_{compression} = \frac{W_{data}}{W_{img}} = \frac{509282}{3256320} = 0,156398019851$

Podemos decir que nuestros datos comprimidos utilizan cerca del **16% del tamaño del archivo original.**

Al obtener el valor promedio de los datos codificados podemos hablar de que ancho de banda

necesitamos para transmitir el video a 30 cuadros por segundo (Calculado en el programa Python):

$bw \approx 15.27846[Mbps]$

Es decir, necesitaríamos un ancho de banda de al menos 15.3 Mbps para poder enviar en tiempo real el video a 30 cuadros por segundo.

Lamentablemente, 15.3 Mbps no está en la lista de posibles velocidades. Es por esto que llegamos a la conclusión de que existe cierto "compromiso" o *trade-off* al momento de priorizar el ancho de banda: O disminuimos la calidad de imágen (utilizar otra matriz Q) o disminuimos la tasa de cuadros por segundo.

Considerando que el video transmitido debe ser de suficiente calidad para poder distinguir incendios forestales, decidimos que no sacrificaríamos la calidad de la imagen, es decir optaremos por encontrar una tasa de cuadros por segundo que se adapte a las velocidades de conexión solicitadas.

Dentro del programa, creamos una tabla que muestra cuanto ancho de banda mínimo se necesitaría para transmitir el video a ciertos fps:

FPS	min Kbps	min Mbps
30	15278.46	15.27846
25	12732.05	12.73205
20	10185.64	10.18564
15	7639.23	7.63923
10	5092.82	5.09282
5	2546.41	2.54641
2	1018.564	1.018564
1	509.282	0.5 09282

Conclusión

A partir de los algoritmos, pruebas y análisis realizados en este trabajo podemos concluir que utilizar la transformada coseno al momento de transformar las imágenes ayuda bastante a la compresión de datos, al aprovechar su ventajosa propiedad de concentrar la mayor cantidad de información en pocos coeficientes, y que gracias a la codificación basada en la distribución de probabilidades de los valores de los datos es posible disminuir considerablemente el tamaño de archivos que se está enviando. Pese a que en esta implementación la parte emisora y receptora comparten el diccionario cabe destacar que en un ejemplo de la vida real el diccionario debería enviarse juntos con los datos codificados. Esto podría aumentar el tamaño de los datos a enviar pero aún así, el porcentaje de los datos codificados enviados seguirá siendo menor.

References

[1]"JPEG 101 - How does JPEG work?". https://arjunsreedharan.org/post/146070390717/jpeg-101-how-does-jpeg-work.

[2]J. R. C. B. Elena Aguilar Fernández, "Decodificador de vídeo MPEG-2 en Matlab y análisis del bitstream".

[3]L.-W. Chang, C.-Y. Wang, and S.-M. Lee, "Designing JPEG quantization tables based on human visual system", in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, 1999.

[4]"Comprimiendo datos - el algoritmo de Huffman en Python". http://bitybyte.github.io/Huffman-coding/.