Lucas Cajal
Grup 8.1

DESCRIPCIÓ DE L'ALGORISME JPEG

Descripció:
L'algorisme JPEG serveix per comprimir amb pèrdues imatges. Aquest algorisme permet ajustar el nivell de compressió (i per tant de fidelitat) a l'hora de comprimir un arxiu.

Aquest algorisme es basa en la idea que els humans percebem molt millor els canvis en la lluminositat d'una imatge que els canvis en el color, per tant es permet reduir més la fidelitat dels canals de color que de la luminància. A més, utilitza la transformada cosinus discreta, ja que s'assumeix que qualsevol senyal numèrica pot ser recreada utilitzant una combinació de funcions de cosinus de diferent freqüència i amplitud.

Compressió:

1. **Conversió de color:** El primer pas és convertir l'espai de color de RBG a YCbCr, on el primer canal Y mostra la lluminositat del píxel, el component Cb representa la diferència entre blau i vermell i Cr la diferència entre el verd i el vermell.

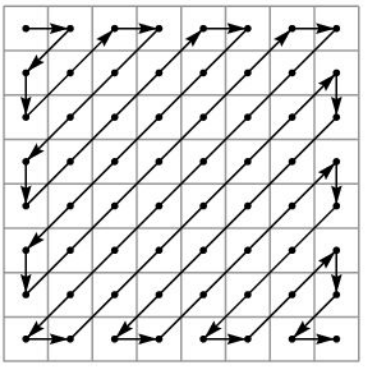
2. **Downsampling [Opcional]:** El següent pas és fer un downsampling als dos canals de crominància, Cb i Cr. Es pot reduir aquesta informació cromàtica a la meitat (4:2:2) o al quart (4:2:0). El més habitual és utilitzar 4:2:0.

3. **Divisió en blocs:** El següent pas és dividir la matriu de píxels de cada canal en blocs de 8x8. En cas que quedin píxels penjats, s'ha de replicar l'últim píxel visitat fins a completar el bloc de 8x8.

4. **DCT:** A partir d'aquí treballem cada bloc per separat. Per a cada un, hem d'aplicar la transformada cosinus discreta número 2 en dues dimensions (DCT-II o simplement DCT).

5. **Quantització:** Amb la matriu resultant, cal dividir-la element per element per la matriu de quantització de luminància en el cas del canal Y i per la matriu de quantització de crominància en els altres dos canals. Aquestes matrius són constants que, realitzant aquesta divisió, ens permeten obtenir números molt propers a zero quan més ens apropem a la cantonada inferior dreta de la matriu original. De forma opcional, es pot multiplicar aquesta matriu per un valor que ens indica la qualitat de l'arxiu comprimit, ampliant o reduint l'efecte que obtenim d'aquesta divisió i per tant forçant l'aparició de més o menys zeros.

6. **Zig-Zag:** Ara cal aplanar aquesta matriu per convertir-la en un vector (de 64 elements). Per fer-ho, anem col·locant cada element de la matriu en el vector seguint l'ordre indicat per un moviment de zig-zag (vegeu figura). Com que la cantonada inferior



dreta té tendència a tenir zeros, els últims elements del vector tendeixen a ser zeros.

7. **Compressió Huffmann:** Per acabar, comprimim aquest vector utilitzant l'algorisme sense pèrdues de Huffmann.

Descompressió:
Per descomprimir, només cal aplicar els passos utilitzats per comprimir però a la inversa:

1. **Descompressió Huffmann:** Primer, es reconstrueix de forma exacta el vector de 64 elements que representa cada bloc.

2. **Zig-Zag:** Es tornen a reorganitzar els elements del vector en una matriu de 8x8 píxels.

3. **Quantització inversa:** El següent pas és fer la quantització inversa: en comptes de dividir element a element la matriu per les matrius constants corresponents, ara multipliquem, utilitzant el mateix factor escalador de qualitat que hem utilitzat en comprimir.

4. **IDCT:** Ara cal aplicar la tercera transformada cosinus discreta a la matriu (DCT-III o IDCT), per intentar reconstruir la matriu original a partir de les diferents funcions de cosinus.

5. **Agrupament dels blocs:** Un cop fetes les transformacions a cada bloc, cal tornar a agrupar-los en una gran matriu per a cada canal.

6. **Upsampling [opcional]:** Si hem utilitzat downsampling per als canals de crominància, cal fer un upsampling per tornar a recuperar la quantitat de píxels que té la imatge original.

7. **Conversió de color:** Per últim, només cal tornar a convertir la imatge del perfil de color YCbCr a RGB.

Estat actual de la implementació de l'algorisme:

Actualment el programa aplica totes les transformacions necessàries a una imatge segons l'algorisme JPEG, excepte la codificació Huffman. D'aquesta ja està implementada la compressió, però cal implementar la descompressió i la persistència per guardar i llegir l'arxiu comprimit. La lectura i escriptura de .PPM està implementada, però cal separar-les de la capa de domini i moure-les a la capa de persistència.

Pel que fa a l'algorisme JPEG en si, s'ha implementat la possibilitat d'escollir el tipus de downsampling a utilitzar (4:4:4, 4:2:2 o 4:2:0) i s'ha paral·lelitzat una part del codi per obtenir una compressió més ràpida, però encara cal aprofundir més en aquest sentit per poder obtenir encara un major rendiment.

Material consultat:

freeCodeCamp: "How jpg works"
https://www.freecodecamp.org/news/how-jpg-works-a4dbd2316f35/

Wikipedia: JPEG
https://en.wikipedia.org/wiki/JPEG

Wikipedia: Discrete cosine transform
https://en.wikipedia.org/wiki/Discrete_cosine_transform

"Image compression and the Discrete Cosine Transform", de Ken Cabeen i Peter Gent
https://www.math.cuhk.edu.hk/~lmlui/dct.pdf

Xilinx: The DCT/IDCT Solution Customer Tutorial
http://homepages.cae.wisc.edu/~ece554/website/Xilinx/app_notes/DCT_IDCT%20Customer%20Tutorial%20custdct.pdf

Mathworks: Compute 2-D discrete cosine transform (DCT)
https://www.mathworks.com/help/vision/ref/2ddct.html

Mathworks: Compute 2-D inverse discrete cosine transform (IDCT)
https://www.mathworks.com/help/vision/ref/2didct.html