Máster Universitario en Ingeniería Informática

PRÁCTICA JPEG

Versión 1. Última actualización 15/11/2021 10:00.

Contenidos

Instrucciones

- La práctica se realiza de forma individual.
- La entrega se realiza a través de la tarea correspondiente en el Campus Virtual (Moodle).
- **La fecha límite de entrega se indica en la tarea de entrega**.
- **No entregar las imágenes comprimidas en ningún caso. Incluir únicamente la imagen original que se carga en el *live script* principal.**
- **Es muy importante que cada práctica sea original**.

Material necesario

- En esta práctica necesitas Matlab con el *toolbox* de procesado de imagen. La licencia campus de Matlab en la UDC está accesible en https://matlab.udc.es

Objetivo

En esta práctica se propone la realización de una implementación básica de un codificador y un descodificador de JPEG. El rendimiento del códec JPEG se evaluará ejecutando el par codificador-descodificador variando el factor de calidad de compresión y, para cada valor de éste, se calcularán las métricas de calidad SSIM y MSE, la tasa de compresión en bit/píxel, el tiempo de codificación y el tiempo de descodificación. Cada métrica se dibujará en una gráfica independiente y se comparará con el resultado proporcionado por la implementación de Matlab con **`imwrite()`**. La Figura 1 y la Figura 2 muestran gráficas de ejemplo para la imagen **`coumpound_eyes.jpg`**. Ten en cuenta que la gráfica depende de la imagen seleccionada.

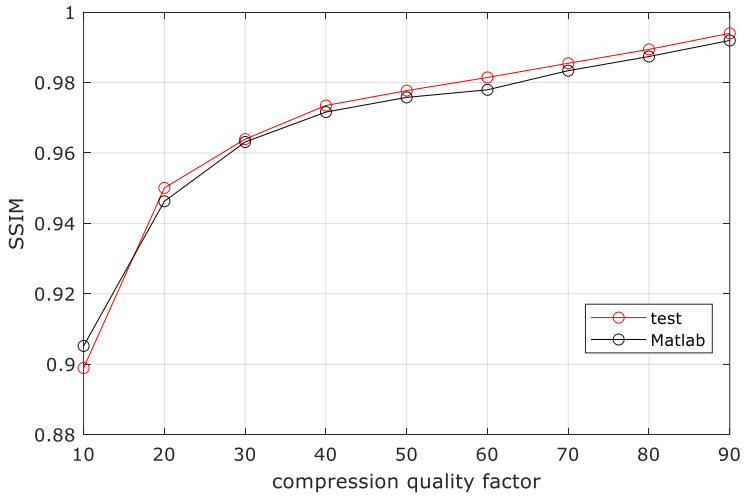

Figura 1: SSIM en función del factor de calidad de compresión para la imagen `coumpound_eyes.jpg`.

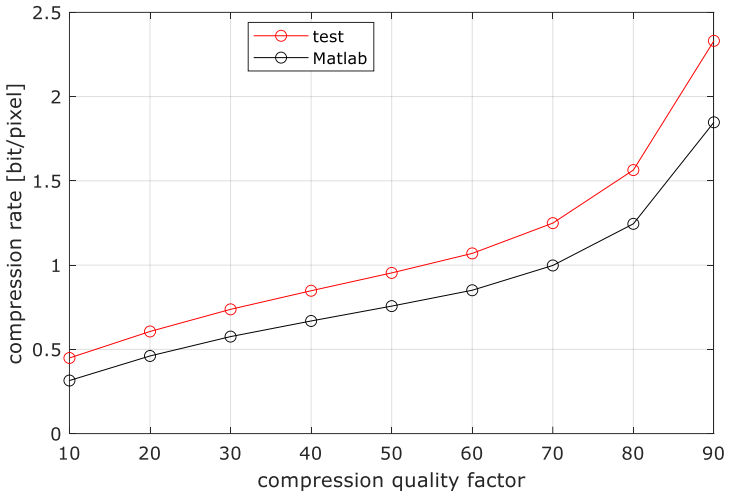

Figura 2: tasa de compresión (expresada en bit/píxel) en función del factor de calidad de la compresión para la imagen `coumpound_eyes.jpg`.

Metodología

La práctica consta de una parte obligatoria y dos opcionales. La idea es obtener, al cabo de tres semanas, un par codificador-descodificador JPEG que soporte las siguientes características:

- El bitstream de la imagen codificada se representará como un array de caracteres (limitaciones de Matlab) y no se guardará en fichero. El bitstream tampoco incluirá las tablas de los códigos Huffman ni las matrices de cuantificación, ya que éstas serán deducidas por el descodificador.
- El descodificador admitirá como entrada la representación del bitstream, las dimensiones de la imagen y el parámetro de calidad utilizado en el codificador.
- Tanto el codificador como el descodificador asumirán imágenes cuyas dimensiones son múltiplos enteros de 8, de forma que la división en bloques de 8×8 píxeles es sencilla. Se proporciona la función **`resizeImageTo8()`** que realiza el preprocesado necesario.

Ten en cuenta que la imagen que devuelve la función `resizeImageTo8()` deberá tomarse como la imagen de referencia para la evaluación de las métricas de calidad (SSIM y MSE).

- Imágenes en escala de grises (el soporte de imágenes en color se incluye en la parte III opcional).
- No se realiza diezmado de crominancias (4:4:4).
- Codificación Huffman para los coeficientes de DC (DPCM en la parte II opcional).
- RLE y codificación Huffman para los coeficientes de AC.
- La codificación Huffman se realiza en base a las tablas definidas en el anexo del estándar.
- Estimación del tamaño en bits de la imagen codificada. La función **`jpegCodedImageSize()`** realiza dicho cálculo a partir del bitstream y las dimensiones de la imagen.

Se recomienda implementar pares de bloques y probarlos antes de pasar al siguiente. El ejemplo para la imagen **`tiny.png`** puede ser de utilidad.

Funciones proporcionadas

Para la realización de la práctica se proporcionan las siguientes funciones:

Estas funciones pueden modificarse y/o mejorarse. Se incluyen a modo de ayuda.

- **`block2zigzag()`** y **`zigzag2block()`**
  - A partir de una matriz 8×8, **`block2zigzag()`** realiza el recorrido en zig-zag tal y como se indica en el estándar JPEG y devuelve el vector resultante. La función **`zigzag2block()`** realiza la operación recíproca.
- **`computeDCTMatrix()`**
  - Devuelve la matriz DCT tipo-II para bloques de 8×8 píxeles.
- **`huffman_codes()`**
  - Implementada por Thomas Holton. Incluye las tablas de códigos Huffman para los coeficientes DC y AC.
- **`huffmanBinaryToDecimal()`** y **`huffmanDecimalToBinary()`**
  - Convierten de representación decimal a binaria y viceversa de acuerdo con los requisitos del código Huffman en JPEG. La representación decimal se realiza como arrays de caracteres 0 o 1 que indican el valor de los bits correspondientes.
- **`jpegCodedImageSize()`**
  - Calcula el tamaño de la imagen en bit/pixel a partir de la representación de su bitstream codificado y sus dimensiones.
- **`jpegQuantizationMatrices()`**
  - Devuelve las matrices de cuantificación para la luminancia y las crominancias.
- **`resizeImageTo8()`**
  - Utiliza la función **`imresize()`** de Matlab para garantizar que las dimensiones de la imagen son múltiplos enteros de 8. La imagen que devuelve esta función **deberá tomarse como la imagen de referencia para las métricas de calidad (SSIM y MSE)**. **Esta operación debe realizarse fuera de la función que implementa el codificador**.
- **`scaleQuantizationMatrix()`**
  - Escala la matriz de cuantificación definida para calidad = 50 de acuerdo con el parámetro de calidad proporcionado. Si calidad = 100, entonces la matriz de cuantificación resultante contiene únicamente 1 (no hay cuantificación), lo que es útil para labores de depuración.

Funciones que deberán ser implementadas

Los prototipos de las funciones que se indican a continuación están incluidos como parte del enunciado de la práctica. Dichos prototipos incluyen documentación que especifica su funcionamiento y que debe ser respetado por la implementación.

- `convertRGBToYCbCr()` y `convertYCbCrToRGB()`
  - Realizan la conversión RGB a YCbCr y viceversa. Su implementación se ha propuesto en la primera semana.
  - Solamente son necesarias si se implementa la parte II opcional.
- `jpegHuffmanEncodeBlock()` y `jpegHuffmanDecodeBlock()`
  - Implementan la codificación (y descodificación) de un bloque de 8×8 píxeles. Realizan las operaciones de codificación DPCM y Huffman para coeficientes DC y RLE y Huffman para los coeficientes AC.
- `jpegEncoder()` y `jpegDecoder()`
  - Implementan la codificación y descodificación JPEG. Los prototipos de las funciones son *live functions* e incluyen documentación detallada de como se debe generar el bitstream.

Ten en cuenta que la implementación que se propone en esta práctica está inspirada en JPEG, pero no sigue el estándar totalmente ni lo implementa de forma completa.

`jpegEncoder()` y `jpegDecoder()` documentan una estructura propuesta para la representación del bitstream basada en un cell array que contiene en su interior un cell array para cada bloque de 8×8 píxeles de la imagen. Cada palabra codificada con el código Huffman se almacena como un par de elementos del cell array para el bloque: el código Huffman de la tabla correspondiente y el valor codificado con la función `huffmanDecimalToBinary()`. Esta estructura simplifica enormemente la implementación del descodificador puesto que es inmediato extraer los elementos del bitstream correspondientes a los bloques de 8×8 píxeles y, dentro de cada bloque, es inmediato obtener el código a buscar en la tabla de códigos Huffman correspondiente.

No es obligatorio utilizar la estructura propuesta para la representación del bitstream.

Live script principal

A partir de las funciones definidas en el apartado anterior (y otras que se considere en la implementación) se deberá implementar un único *live script*, cuyo prototipo se incluye en el fichero `IGM_Practica_JPEG.mlx` y que contendrá el código, los resultados y **sus comentarios**.

El *live script* principal evaluará la SSIM, el MSE, la tasa de compresión en bit/píxel y los tiempos de codificación y descodificación de la implementación para los valores del parámetro de calidad de compresión `10:10:90` y los comparará con los resultados producidos por la implementación de Matlab.

La evaluación se realizará con una imagen disponible en internet a seleccionar por parte del estudiante.

Se indican a continuación las tareas a realizar por el *live script*:

- Carga de la imagen original.
- Ajuste de sus dimensiones para que sean múltiplos enteros de 8 usando **`resizeImageTo8()`**.
- Inicialización de los vectores del parámetro de calidad y de los que se usarán para guardar los resultados de las métricas.
- Bucle que itera por todos los valores del parámetro de calidad de compresión:
    - Compresión mediante **`jpegEncoder()`** y medición de tiempo.
    - Descompresión mediante **`jpegDecoder()`** y medición de tiempo.
    - Compresión y descompresión con **`imwrite()`** e **`imread()`** y medición de tiempos.
    - Cálculo de los valores de las métricas para el códec implementado y para el de Matlab.
    - Dibujo de la imagen original y la imagen comprimida con calidad 90.
    - Dibujo de las gráficas de SSIM, MSE, tasa de compresión en bit/s, tiempo de codificación y tiempo de descodificación, todas con respecto al parámetro de calidad de compresión.
        - Las gráficas **deben comentarse debajo de cada una de ellas**.

Opciones de entrega

La práctica consta de una parte obligatoria y dos opcionales. Al inicio del *live* script principal se indicarán las partes implementadas. Las funcionalidades de cada parte se especifican a continuación. Ten en cuenta que el orden de las partes opcionales es meramente informativo.

Parte I obligatoria [7 puntos]

- Codificación y descodificación de imágenes en **escala de grises** cuyas dimensiones son múltiplos de 8.
- Preprocesado: conversión de los valores de píxel a **`double`**.
- División de la imagen en bloques de 8×8 píxeles.
- Para cada bloque:
    - Aplicación de la DCT en 2D.
    - Cuantificación de los coeficientes transformados.
    - Conversión del bloque de 8×8 píxeles en un vector de 64 elementos.
    - Codificación Huffman del vector mediante **`jpegHuffmanEncodeBlock()`**:
        - Coeficientes DC se codifican mediante las tablas correspondientes **sin aplicar DPCM**.
        - Coeficientes de AC se procesan primero mediante RLE (rachas de ceros) y, a continuación, se codifican mediante las tablas correspondientes. Atención a las secuencias que dan lugar a códigos ZRL.
- Cada bloque finaliza con el código EOB correspondiente a la luminancia.
- El descodificador deshace las operaciones anteriores y convierte el resultado al tipo **`uint8`** de forma que pueda ser visualizado con **`imshow()`**.
- Estimación del tamaño que ocupa la imagen mediante la función **`jpegCodedImageSize()`**.

Parte II opcional [1 punto]

- Se añade la codificación DPCM a los coeficientes de DC cuantificados antes de entrar al codificador Huffman.

Parte III opcional [2 puntos]

- Soporte de imágenes en color.
  - Conversión de RGB a YCbCr utilizando las funciones **`convertRGBToYCbCr()`** y **`convertYCbCrToRGB()`**.
  - Las funciones **`jpegHuffmanEncodeBlock()`** y **`jpegHuffmanDecodeBlock()`** tienen que actualizarse para soportar las tablas de códigos Huffman para las crominancias. Los códigos EOB y ZRL también son distintos para las crominancias.

Valoración

Se valorarán los siguientes aspectos:

- Corrección de los resultados obtenidos y de la implementación realizada. Para reforzar los comentarios, se recomienda hacer pruebas con diferentes imágenes con distintas características.
- **Calidad del código** (nombrado de variables, estructuración, etc.) y de la documentación del código. En general, el código debe ser fácil de leer y entender.
- **Concisión** y **calidad** de los comentarios de los resultados obtenidos. Ten en cuenta que se pueden realizar ejecuciones y mostrar imágenes de resultado, no sólo las gráficas pedidas, para reforzar los comentarios realizados.
- **En los comentarios trata de evitar opiniones y/o conjeturas**.
- **El objetivo de la práctica no es aproximarse a los resultados que se obtienen con la implementación de Matlab disponible en `imwrite()`**. Que la ejecución sea más o menos lenta o que la tasa de compresión sea mayor o menor no es relevante excepto, claro está, que estemos hablando de diferencias exageradas:
  - Por ejemplo, si para una imagen y valor del factor de calidad de la compresión Matlab arroja un resultado de 1 bit/píxel y nuestra implementación necesita más de 10 bit/píxel es indicativo de que hay algún problema en el código.
  - Si la gráfica de la tasa de compresión oscila ostensiblemente a medida que aumenta el factor de la calidad de la compresión también es síntoma de algún problema.

Rúbrica de evaluación

Parte I	0 puntos	0,3 puntos	0,5 puntos
Parte I: preprocesado y postprocesado de la imagen.	Hay errores importantes, como por ejemplo usar las funciones im2double() e im2uint8(), o faltan pasos fundamentales.	Hay algún error de importancia menor.	Está correcto.
Parte I: Aplicación de la DCT y la DCT inversa a bloques 8x8 mediante multiplicación de matrices.	No se aplica correctamente o hay errores graves.	Hay algún problema menor con la implementación de la DCT y/o su inversa o se realiza de forma ineficiente, como por ejemplo calculando la matriz para cada bloque.	La implementación es correcta y eficiente.
Parte I: Aplicación de la cuantificación y la cuantificación inversa a bloques 8x8 mediante multiplicación de matrices.	No se aplica correctamente o hay errores graves, como por ejemplo que el resultado de la cuantificación no sea entero.	Hay algún problema menor con la implementación de la cuantificación y/o su inversa o se realiza de forma ineficiente, como por ejemplo calculando la matriz para cada bloque.	La implementación es correcta y eficiente.
Parte 1: serialización / deserialización de los bloques de 8x8 píxeles de la imagen	No se realiza la serialización.	La serialización no es la definida en el estándar.	Se aplica la serialización correctamente mediante las funciones blockToZigzag() y zigzagToBlock().
Parte I: codificación Huffman de coeficientes DC.	No se realiza la codificación o hay errores graves en su implementación.	No se implementa adecuadamente la codificación Huffman cuando el valor de DC es cero o hay algún otro error menor.	La implementación es correcta y eficiente.
Parte I: codificación Huffman de coeficientes AC	No se realiza la codificación o hay errores graves en su implementación.	No se generan correctamente los códigos ZRL cuando hay rachas de ceros. No se codifican correctamente las rachas de ceros usando la tabla de códigos Huffman para coeficientes AC. Se generan códigos ZRL innecesarios porque aparecen justo antes de EOB sin ningún coeficiente no cero antes.	La implementación es correcta y eficiente.
Parte I: representación del bitstream.	El bitstream se genera de forma incorrecta y/o es incoherente con la implementación del descodificador.	Hay algún error en la generación del bitstream.	El bitstream es correcto y coherente con la implementación del descodificador.
Parte I:	La estimación es errónea o se utilizan métodos erróneos, como por ejemplo volver a escribir la	Hay algún error en la estimación, pero no es crítico.	La estimación es correcta.

Estimación del tamaño que ocupa la imagen codificada	imagen reconstruida mediante imwrite().		
Parte I: calidad del código	El código es muy difícil de entender, las variables no siguen ningún tipo de criterio de nombrado.	Hay algunos problemas de estructuración del código, como por ejemplo no incluir todas las operaciones del codificador y descodificador en las funciones pedidas.	El código es de calidad y se entiende fácilmente.
Parte I: Resultados	Los resultados obtenidos para SSIM, MSE, tasa de compresión y/o tiempos de codificación y descodificación no son correctos en absoluto.	Alguno de los resultados no es correcto.	Todos los resultados son correctos.
Parte 1: Comentarios de los resultados	No hay ningún tipo de comentario.	Faltan comentarios en alguno de los resultados o son erróneos.	Todos los resultados están comentados.

Parte II	**0 puntos**	**0,3 puntos**	**0,5 puntos**
Parte II: Codificación /descodificación DPCM de las componentes de DC	No se ha implementado.	Se ha implementado, pero hay algún error en la implementación.	La implementación es correcta. Los coeficientes DC y AC se separan correctamente en el codificador, se realiza la DPCM y se vuelven a unir en los bloques en el descodificador después de descodificar DPCM.

Parte III	**0 puntos**	**0,3 puntos**	**0,5 puntos**
Parte III: Conversión a YCbCr en el codificador y a RGB en el descodificador.	No se implementa.	La conversión es defectuosa o hay algún error en su implementación.	La implementación es correcta.
Parte III: codificación Huffman de coeficientes DC.	No se realiza o no se tienen en cuenta las tablas para las crominancias.	Hay algún error en la implementación.	La implementación es correcta y eficiente.
Parte III: codificación Huffman de coeficientes AC	No se realiza o no se tienen en cuenta las tablas para las crominancias.	Los códigos ZRL y/o EOB no se generan adecuadamente para bloques de luminancia y crominancia.	La implementación es correcta y eficiente.
Parte III: representación del bitstream	El bitstream es incorrecto al añadir las crominancias.	Hay algún error en la representación del bitstream.	El bitstream es correcto y coherente con la implementación del descodificador.
Parte III: Integración con el código de la parte I.	No se integra la implementación de la parte III en el código de la parte I, sino que se utiliza código completamente distinto.	Se integra parcialmente con el código de la parte I.	Se integra completamente con el código de la parte I.
Parte III: Estimación del tamaño que ocupa la imagen codificada.	No se implementa o la estimación es errónea de forma que el tamaño estimado no permite interpretar correctamente los resultados.	Hay algún error en la estimación, pero no es crítico.	La estimación es correcta.
Parte III: Resultados	No se implementa o los resultados obtenidos para SSIM, MSE, tasa de compresión y tiempos de codificación y descodificación no son correctos en absoluto.	Alguno de los resultados no es correcto.	Todos los resultados son correctos.
Parte III: Comentarios	No hay ningún tipo de comentario.	Faltan comentarios en alguno de los resultados o son erróneos.	Todos los resultados están comentados.