Learning Objectives

- Be able to work with and JPEG images and other representations in the transform domain.
- Understand what happens during JPEG compression, and its potential consequence to watermarking and steganography.
- Be able to apply simple LSB embedding in the JPEG domain.

Overview

The elements of JPEG

- Operates on luminence and chrominance (YCbCr) (not on RGB)
  - Grayscale images have luminence component only.
- Downsampling
- Works in the DCT domain (not the spatial domain)
- Quantisation
- Entropy coding (lossless compression)

JPEG is not a file format

- JPEG is a compression system
  - The system employs three different compression techniques
- JPEG is not a file format.
- Files with extension .jpeg are often JFIF or EXIF.
  - JFIF is traditionally the most common file format for JPEG.
  - EXIF is made for digital cameras and contain extra meta information.
Core Reading

**Digital Image Processing Using MATLAB.**
- Chapter 6: colour images
  - Representation
  - Processing
  - Conversion
- Chapter 8.5: JPEG compression
- Chapter 4: Frequency domain processing

### Alternatives to RGB

- **NTSC:** \((Y, I, Q)\)
  \[
  \begin{bmatrix}
  Y \\
  I \\
  Q
  \end{bmatrix} =
  \begin{bmatrix}
  0.299 & 0.587 & 0.114 \\
  0.596 & -0.274 & -0.322 \\
  0.211 & -0.523 & 0.312
  \end{bmatrix}
  \begin{bmatrix}
  R \\
  G \\
  B
  \end{bmatrix}
  \]
  where \(R, G, B \in [0, 1]\).

- **YCbCr:** \((Y, Cb, Cr)\)
  \[
  \begin{bmatrix}
  Y \\
  Cb \\
  Cr
  \end{bmatrix} =
  \begin{bmatrix}
  16 \\
  128 \\
  128
  \end{bmatrix}
  +
  \begin{bmatrix}
  65.481 & 128.553 & 24.966 \\
  -37.797 & -74.203 & 112.000 \\
  112.000 & -93.786 & -18.214
  \end{bmatrix}
  \begin{bmatrix}
  R \\
  G \\
  B
  \end{bmatrix}
  \]
  where \(R, G, B \in [0, 1]\) and \(Y, Cb, Cr \in [0, 255]\).
The DCT transform

- Several different DCT transforms.
- We use the following.

\[ T_f(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \sqrt{\alpha(u)\alpha(v)} \cos \frac{(2x+1)u\pi}{2M} \cos \frac{(2y+1)v\pi}{2N} \]

where

\[ \alpha(a) = \begin{cases} 
1, & \text{if } a = 0, \\
2, & \text{otherwise.}
\end{cases} \]

\[ M = N = 8, \]

Matlab functions
- `dct2` (2D DCT transform)
- `idct2` (Inverse)
- `blkproc (X, [M N], FUN)`

For instance
- `blkproc (X, [8 8], @dct2)`

Do not use `imread` for JPEG images
- converts images to the spatial domain.
- you don’t even know the compression parameters...

The DCT domain and JPEG

Transform image
**What is sampling?**

**Fact**

The human eye is more sensitive to changes in luminance than in chrominance.

- To sample is to collect measurements.
  - Each pixel is a sample (measuring the colour of the image).
  - Lower resolution means fewer samples.
- Reducing resolution = downsampling
- Basic $M \times N$ image: $N \times M$ samples per component ($Y, Cb, Cr$).
- $Y$ is more useful than $Cb$ and $Cr$.
- Therefore we can downsample $Cb$ and $Cr$
  - $M/2 \times N/2$ is common for $Cb$ and $Cr$
  - Still use $M \times M$ for $Y$

---

**What do we save?**

- Original: $M \times N$ pixels $\times 3$ components.
- Compressed: $2 \times \frac{M}{2} \times \frac{N}{2} + M \times N = 1.5 \frac{M \times N}{2}$
- Ratio
  $$\frac{\text{Compressed}}{\text{Original}} = \frac{1.5MN}{3MN} = \frac{1}{2}.$$  
- We just saved 50%

---

**Chrominance versus Luminence**

**Fact**

The human eye is more sensitive to changes in luminance than in chrominance.

- Watermarking tend to embed in $Y$ (luminence)
- Embedding in $Cb$ and $Cr$ would more easily be destroyed by JPEG

---

**Downsampling in JPEG**

- Translation to YCbCr.
- Downsampling
- DCT transform
  - Each downsampled component matrix $Y, Cb, Cr$ is
    - Divided into $8 \times 8$ blocks
    - DCT transformed blockwise
  - An $8 \times 8$ block in $Cb$ can be associated with $1, 2$ or $4$ $Y$ blocks depending on downsampling.
What is quantisation?

Rounding in general

- Rounding numbers is quantisation.
- Measuring gives continuous numbers
  - Whether you measure pixel luminence, or the length of your garage.
  - No matter how close to points are, there is a point in between.
- However, our precision is limited.
  - We give lengths to the nearest unit.
  - Luminence is categorised into 256 intervals (8bit integers).
- Computer memory is finite, 256 different possibilities for a byte.

Quantisation in JPEG

- Quantisation in the DCT domain
- Each coefficient is divided by the quantisation constant.
- The result is rounded to nearest integer.
- Different quantisation constants for each coefficient in the block.

Example

Quantisation in JPEG

```
DCT matrix
-415  30  -61  27  56  -20  -2  0
 4  -22  -61 10  13  -7  -9  5
 47  77  -25 -29 10  5  -6
 49 12 34 -15 -10 6  2  2
12 7  -13  4  -2  2  3
-1 0 5  -6  -2  1  4  2
-8 3  2  -6  -2  1  4  2
-1 0 5  -2  -1  -3  4  -1
0 0 -1  -4  -1  0  1  2
```

Quantisation matrix

```
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 79 87 103 121 120 101
72 92 95 98 112 150 103 99
```

Quantised DCT matrix

```
-26 -3  -6  2  2  -1  0  0
 0  -2  -4  1  1  0  0  0
-3 1  5  -1  -1  0  0  0
-4 1  2  -1  0  0  0  0
1 0  0  0  0  0  0  0
0 0  0  0  0  0  0  0
0 0  0  0  0  0  0  0
0 0  0  0  0  0  0  0
```

Example from Wikipedia.
In order to compress the data:
- Use few bits (short codewords) for frequent symbols
- Many bits (long codewords) only for rare symbols
- Usually, JPEG uses a simple Huffman code.
  - it can use other codes (saving space, but computationally costly)
- For instance, a single short codeword to say
  - ‘the rest of the block is zero’

**Steganography in JPEG**

**Fragility of LSB**

- LSB embedding is criticised for being fragile
- JPEG removes insignificant information
  - ... such as the LSB
- JPEG compression after embedding (probably) ruins the message

- When is this a problem?

**Before or after compression**

- It is a problem in robust watermarking
  - JPEG compression is common-place
  - most applications need robustness
- If the purpose is steganography,
  - and Alice and Bob are allowed to exchange pixmaps,
  - then it is not a problem.
- Obviously, if your steganogram is supposed to be JPEG
  - ... **Do not do LSB in the pixmap.**
### Double compression

- Common bug in existing software
- Read an arbitrary image file
  - JPEG is decompressed on reading
  - ... → pixmap
- Embedding works on JPEG
  - image is compressed to produce JPEG signal
  - quality factor (QF) either default or supplied by user
- A JPEG steganogram has now been compressed twice
  - different QF produces an artifact
- Is there any reason for de- and recompressing?

### Important lessons

- Do not make unnecessary image conversions.
- Many techniques apply to any format
  - LSB applies to JPEG signals
  - ... but it is called Jsteg
- Use a technique which fits the target (stego-) format
  - i.e. the format you are allowed to use on the channel.

### Main development

The past at a glance

Core Reading


- JSteg was published
- JSteg was broken
- OutGuess was published
- OutGuess was broken
- F5 was published
- F5 was broken

### The JSteg algorithm

- JSteg denotes a software package.
- Approach: simple LSB in the DCT domain.
- No different from LSB in the spatial domain
- $\chi^2$ analysis applies
Pseudocode

The JSteg algorithm

**Input:** Image $I$, Message $\vec{m}$

**Output:** Image $J$

for each bit $b$ of $\vec{m}$

  $c := $ next DCT coefficient from $I$

  while $c = 0$ or $c = 1$,

    $c := $ next DCT coefficient from $I$

  end while

  $c := c - c \mod 2 + b$

  replace coefficient in $I$ by $c$

end for

- May ignore high and/or low frequency coefficients

Outguess 0.1

**How can we improve JSteg?**

- How did we improve LSB in the Spatial Domain?

**Solution**

*Choose random coefficients from the entire image.*
Pseudocode

Outguess 0.1

**Input:** Image $I$, Message $\vec{m}$, Key $k$

**Output:** Image $J$

Seed PRNG with $k$

**for** each bit $b$ of $\vec{m}$

\[
c := \text{pseudo-random DCT coefficient from } I
\]

**while** $c = 0$ or $c = 1$,

\[
c := \text{pseudo-random DCT coefficient from } I
\]

**end while**

\[
c := c - c \mod 2 + b
\]

replace coefficient in $I$ by $c$

**end for**

- May ignore high and/or low frequency coefficients

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 37 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 38 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 40 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 41 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 42 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 43 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 44 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 45 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 46 / 47

---

**Dr Hans Georg Schaathun**

The DCT domain and JPEG

Spring 2009 – Week 3 47 / 47
w/o the toolbox
Load/Save workspace

- `jpeg_read` and `jpeg_write` are compiled functions,
  - Available on web page for Intel/Linux and Intel/Windows
  - Compilation required for other systems.

- If you have trouble with the toolbox, it is possible to load/save files
  as Matlab workspaces.

```matlab
» load 'image.mat'
» whos
```

```
Name      Size  Bytes  Class  Attributes
im        1x1     575836 struct
```

- There is one mat-file for each image on the web page

Other toolbox functions

- If you want to write your own (de)compressor, the following functions are useful.
  - `bdct`, `ibdct`, `quantize`, `dequantize`

Pseudo-random number generators

- A pseudo-random number generator (PRNG)
  - finite state machine
  - starting state given by user input — `the seed`
  - each transition outputs a number
  - observing a sequence of number,
    - it is computationally infeasible to predict the next number
  - looks random

- It is deterministic
  - if sender and receiver share the seed (a key)
    - they can generate the same sequence

Random locations

- Many ways to do it
  - but some makes it hard to avoid using the same pixel twice

- A simple approach
  - randomly permute the pixels
  - embed in the first pixels in the permuted sequence
  - reverse the permutation

- If the sender and receiver use the same PRNG seed,
  - they get the same pseudo-random permutation
Matlab functions

- Many functions using a PRNG
  - `rand`, `randperm`, `randn`
- And one dedicated for *performing* random permutations
  - `new = randintrlv ( old, seed )`
  - the seed is given directly when the permutation is used
  - `reverse: old = randdeintrlv ( new, seed )`
- `new/old` should be a 1D vector
  - serialise the image into a vector (`im(:)`) before `randintrlv`
  - `reshape ( vector, [ N, M ] )` to recover the 2D image after `randdeintrlv`

Masking

- Matlab allows Boolean (logical) matrices as indices
  - Say $A$ is an $n \times m$ matrix
  - $B = \text{logical}(\text{ones}(n,m))$
  - $B(1,1) = 0$
  - $A(B)$ will be all elements of $A$ except (1,1)
- `repmat(B,m,n)` replicates $B$ to form a larger matrix
  - $m$ times the height
  - $n$ times the width
- How do you extract the AC coefficients in a JPEG image?
  - Make an $8 \times 8$ mask of ones
  - Set the DC entry to 0, e.g. (1,1)
  - Replicate the mask using `repmat`