The DCT domain and JPEG
CSM25 Secure Information Hiding

Dr Hans Georg Schaathun
University of Surrey
Spring 2008

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.
Overview

Reading

Core Reading

*Digital Image Processing Using MATLAB.*

- Chapter 6: colour images
  - Representation
  - Processing
  - Conversion
- Chapter 8.5: JPEG compression
- Chapter 4: Frequency domain processing

The RGB colour representations

- RGB: A colour is a vector \((R, G, B)\)
  - \(R\) is amount of red light.
  - \(G\) is amount of green light.
  - \(B\) is amount of blue light.
- Each pixel can be either
  - A colour vector \((R, G, B)\); or
  - a reference to an array of colour vectors (the palette)
- Each coefficient can be
  - \(\in [0, 1]\); floating point (double in MATLAB)
  - \(\in \{0, 1, \ldots, 255\}\); 8-bit integer (uint8 in MATLAB)
  - \(\in \{0, 1, \ldots, 2^{16} - 1\}\); 16-bit integer (uint16 in MATLAB)

Image Representation

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 \(Y, Cb, Cr \in [0, 255]\).

Block-wise

- Each colour-channel (Y,Cb,Cr) considered separately
- \(M \times N\) matrix divided into 8 \(
\times 8\) blocks
- Each block is handled separately
The DCT transform

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

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

where

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

\[ M = N = 8, \]

Matlab

- Matlab functions
  - dct2 (2D DCT transform)
  - idct2 (Inverse)
  - blkproc (X, [M N], FUN)
- For instance
  - blkproc (X, [8 8], @dct2)
- Use help system for details
- Unfortunately, we do not have the JPEG toolbox.
  - Loading JPEG images converts them to the spatial domain.

The DCT transform

- The inverse is similar

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

where

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

\[ M = N = 8, \]

Transform image

- Linear combination of patterns (see right)
- DC (upper left) gives average colour intensity
- Low frequency: coarse structure
- High frequency: fine details
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: $N \cdot M$ 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:
  $$2 \times \frac{M}{2} \times \frac{N}{2} + M \times N = \frac{1}{2}M \times N$$
- Ratio
  $$\frac{\text{Compressed}}{\text{Original}} = \frac{\frac{1}{2}MN}{3MN} = \frac{1}{2}$$
- 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 downsampling 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

<table>
<thead>
<tr>
<th>DCT matrix</th>
<th>Quantisation matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-415\ -30\ -61\ 27\ 56\ -20\ -2\ 0]</td>
<td>[16\ 11\ 10\ 16\ 24\ 40\ 51\ 61]</td>
</tr>
<tr>
<td>[-47\ 7\ 77\ -25\ -29\ 10\ 5\ -6]</td>
<td>[12\ 12\ 14\ 19\ 26\ 26\ 58\ 60]</td>
</tr>
<tr>
<td>[-49\ 12\ 34\ -15\ -10\ 6\ 2\ 2]</td>
<td>[14\ 13\ 16\ 24\ 40\ 57\ 69\ 56]</td>
</tr>
<tr>
<td>[-12\ -7\ -13\ -4\ -2\ 2\ -3\ 3]</td>
<td>[14\ 17\ 22\ 29\ 51\ 87\ 80\ 62]</td>
</tr>
<tr>
<td>[-8\ 3\ 2\ -6\ -2\ 1\ 4\ 2]</td>
<td>[18\ 22\ 37\ 56\ 68\ 109\ 103\ 77]</td>
</tr>
<tr>
<td>[-1\ 0\ 0\ -2\ -1\ -3\ 4\ -1]</td>
<td>[24\ 35\ 55\ 64\ 81\ 104\ 113\ 92]</td>
</tr>
<tr>
<td>[0\ 0\ -1\ -4\ -1\ 0\ 1\ 2]</td>
<td>[49\ 64\ 79\ 87\ 103\ 121\ 120\ 101]</td>
</tr>
</tbody>
</table>

Quantised DCT matrix

<table>
<thead>
<tr>
<th>Quantised DCT matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-26\ -3\ -6\ 2\ 2\ -1\ 0\ 0]</td>
</tr>
<tr>
<td>[0\ -2\ -4\ 1\ 1\ 0\ 0\ 0]</td>
</tr>
<tr>
<td>[-3\ 1\ 5\ -1\ -1\ 0\ 0\ 0]</td>
</tr>
<tr>
<td>[-4\ 1\ 2\ -1\ 0\ 0\ 0\ 0]</td>
</tr>
<tr>
<td>[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]</td>
</tr>
<tr>
<td>[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]</td>
</tr>
<tr>
<td>[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]</td>
</tr>
</tbody>
</table>

Example from Wikipedia.

Entropy Coding

- Recall
  ```
  \[-26\ -3\ -6\ 2\ 2\ -1\ 0\ 0\] 
  \[0\ -2\ -4\ 1\ 1\ 0\ 0\ 0\] 
  \[-3\ 1\ 5\ -1\ -1\ 0\ 0\ 0\] 
  \[-4\ 1\ 2\ -1\ 0\ 0\ 0\ 0\] 
  \[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\] 
  \[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\] 
  \[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\]
  ```

- Observe
  - 0 is extremely common
  - ±1 is common
  - Two-digit numbers are very rare

- This is typical
Entropy Coding

- 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’

Before or after compression

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?

- 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
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
- $c := c \mod 2 + b$
- replace coefficient in $I$ by $c$

end for

- May ignore high and/or low frequency coefficients

---

Pros and Cons

- **Why is JSteg important?**
  - First publicly available solution.
  - Simple solution

- **What are the disadvantages?**
  - Similar to LSB in Spatial domain.
  - Histogramme analysis applies
  - $\chi^2$/pairs of values applies

---

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

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

replace coefficient in \( I \) by \( c \)

end for

* May ignore high and/or low frequency coefficients

---

**Histogramme analysis**

- Pairs of values
  - Generalised \( \chi^2 \) works

- Symmetry
  - Embedding exchange \( +2 \leftrightarrow +3 \) and \( -2 \leftrightarrow -1 \).
  - The DCT histogramme is expected to be symmetric
  - Outguess/JSteg destroy the symmetry.

**Symmetry analysis**

\[
J_{2i} = \left(1 - \frac{q}{2}\right)I_{2i} + \frac{q}{2}I_{2i+1},
\]

\[
J_{2i+1} = \frac{q}{2}I_{2i} + \left(1 - \frac{q}{2}\right)I_{2i+1}
\]

Writing

\[ a = \frac{1 - q/2}{1 - q} \quad \text{and} \quad b = \frac{q/2}{1 - q}, \]

we get

\[ I_{2i} = aJ_{2i} - bJ_{2i+1}, \]

\[ I_{2i+1} = -bJ_{2i} + aJ_{2i+1}. \]

In other words

\[
\sum_{i>0} I_{2i} = \sum_{i<0} I_{2i}, \quad \text{and} \quad \sum_{i>0} l_{2i+1} = \sum_{i<0} l_{2i+1}.
\]

- Substitute for \( l_{2i} \) and \( J_{2i+1} \)
- One equation in one unknown : \( q \)
Exercise
Symmetry analysis

\[ \sum_{i>0} l_{2i} + \sum_{i<0} l_{2i+1} = \sum_{i<0} l_{2i} + \sum_{i>0} l_{2i+1}. \]

- Substitute for \( l_{2i} \) and \( l_{2i+1} \)
  - get equation in \( J_{2i} \) and \( J_{2i+1} \).
- Solve for \( J_1 \), write as equation in
  - \( J_1 \) (left hand side)
  - \( \Delta_i = J_{2i} - J_{2i+1} \) (\( i > 0 \)), and \( \Delta_i = J_{2i+1} - J_{2i} \) (\( i < 0 \)).
- Solve for \( q \)
- Can you implement the resulting expression for \( p \) in Matlab?

Improvements at a glance

- F3 – F4 changes the embedding
  - better histogramme – maintain symmetry
- Statistics-aware embedding
  - Outguess 0.2 uses unused capacity
  - dummy changes are used to even out the statistics
- Matrix coding/wet paper coding
  - F5 minimises distortion using coding theory
  - fewer bit-flips per message bit
  - techniques from coding for restricted memory

The problem of F3

- Reembedding
  - Zero created by message zeroes
  - Zeros are reembedded
  - Extra zeros embedded \( \Rightarrow \) overweight of even coefficients.

Bitflips in F3

- Avoid pairs of values.
- Always decrease absolute value when changing
- Zeros are ignored.
- Zero created: reembed in new coefficient

Typical JPEG.

Typical F3.
**F4 : evening out even values**

- Swap interpretation for negative coefficients
- +1 causes reembedding of zero
- -1 causes reembedding of one

<table>
<thead>
<tr>
<th>Cover</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+4</td>
<td>+5</td>
</tr>
<tr>
<td>Stego</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+4</td>
<td>+5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cover</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+4</td>
<td>+5</td>
</tr>
<tr>
<td>Stego</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+4</td>
<td>+5</td>
</tr>
</tbody>
</table>

**F4 histogramme**

- What does the histogramme look like after F4 embedding?
- Coefficients have generally been decreased.
- Effect similar to reduced quality factor.
- F4 (and F5) can be broken

*Suggested Reading*

**Example**

*Matrix embedding*

- \( n = 3 \) coefficients (LSB): \( x_1, x_2, x_3 \)
- \( k = 2 \) message bits: \( m_1, m_2 \)

**Decoder**
- \( \hat{m}_1 = x_1 \oplus x_3 \)
- \( \hat{m}_2 = x_2 \oplus x_3 \)

**Encoder**
- \( m_1 = x_1 \oplus x_3 \) and \( m_2 = x_2 \oplus x_3 \) : no change
- \( m_1 \neq x_1 \oplus x_3 \) and \( m_2 = x_2 \oplus x_3 \) : change \( x_1 \)
- \( m_1 = x_1 \oplus x_3 \) and \( m_2 \neq x_2 \oplus x_3 \) : change \( x_2 \)
- \( m_1 \neq x_1 \oplus x_3 \) and \( m_2 \neq x_2 \oplus x_3 \) : change \( x_3 \)

Average number of changes: \( 3/4 \) per two bits
- Compare to F4 (and others): 1 per two bits
Matrix coding

- Longer codewords (larger $n$) saves more
- e.g. $k = 9$; $n = 511$: change $\approx 1/9$ pixels per message bit
- Matrix embedding based on coding theory
- We will return to coding theory and matrix coding later

Statistics-aware embedding

- Modify unused coefficients
  - Mimick original statistic
  - Tune for every statistic used in known analysis techniques
- OutGuess 0.2
  - Embedding as OutGuess 0.1.
  - Modify selected unused coefficients to correct histogram

Combinations

- Statistics-aware Matrix Coding
  - More advanced code allows a choice of coefficient to modify
  - Use choice to mimick original statistical distribution
- Known as Wet Paper Codes
  - Based on Dirty Paper Codes by Costa 1983.

JPEG Toolbox

Reading a JPEG image

```matlab
im = jpeg_read ( 'Kerckhoffs.jpg' )
im =
  image_width: 180
  image_height: 247
  image_components: 3
  image_color_space: 2
  jpeg_components: 3
  jpeg_color_space: 3
  comments: {}
  coef_arrays: {[248x184 double] [128x96 double] [128x96 double]}
  quant_tables: {[8x8 double] [8x8 double]}
  ac_huff_tables: [1x2 struct]
  dc_huff_tables: [1x2 struct]
  optimize_coding: 0
  comp_info: [1x3 struct]
  progressive_mode: 0
```
The JPEG toolbox

JPEG data

» Xy = im.coef_arrays{im.comp_info(1).component_id};
» whos Xy
Name      Size  Bytes  Class  Attributes
Xy  248x184 365056  double

» Q = im.quant_tables{im.comp_info(1).quant_tbl_no}
Q =

  6  4  4  6  10  16  20  24
  5  5  6  8  10  23  24  22
  6  5  6 10  16  23  28  22
  6  7  9 12  20  35  32  25
  7  9 15 22  27  44  41  31
 10 14 22 26  32  42  45  37
 20 26 31 35  41  48  48  40
 29 37 38 39  45  40  41  40

Dr Hans Georg Schaathun
The DCT domain and JPEG
Spring 2008 60 / 62

The JPEG toolbox

w/o the toolbox

Load/Save workspace

• jpeg_read and jpeg_write are compiled functions,
  • Non-trivial to install
• In the exercises, can load presaved workspaces
  » 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

• I recommend that you use the other toolbox functions
  • bdct, ibdct, quantize, dequantize
• These are m-files, which can be copied into current directory.
• Alternatively, use matlab on tweek which has the toolbox.