15-499: Algorithms and Applications

Data Compression IV - Lossy Compression
- Scalar and Vector Quantization
- Transform coding
  - Block cosine (JPEG, MPEG)
  - Wavelets (JPEG2000)

Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, ...
Information Theory: Entropy, etc.
Probability Coding: Huffman + Arithmetic Coding
Applications of Probability Coding: PPM + others
Lempel-Ziv Algorithms: LZ77, gzip, compress, ...
Other Lossless Algorithms: Burrows-Wheeler
Lossy algorithms for images: JPEG, MPEG, ...
  - Scalar and vector quantization
  - JPEG and MPEG
Compressing graphs and meshes: BBK

Scalar Quantization

Quantize regions of values into a single value:

Can be used to reduce # of bits for a pixel

Vector Quantization

In

Generate Vector

Find closest code vector

Encode

Index

Codebook

Out

Generate Output

Index

Codebook

Decode
Vector Quantization

What do we use as vectors?
- Color (Red, Green, Blue)
  - Can be used, for example, to reduce
    24 bits/pixel to 8 bits/pixel
  - Used in some terminals to reduce data rate
    from the CPU (colormaps)
- K consecutive samples in audio
- Block of K pixels in an image
  How do we decide on a codebook
- Typically done with clustering

Linear Transform Coding

Want to encode values over a region of time or space
- Typically used for images or audio
Select a set of linear basis functions $\phi$ that span the space
- sin, cos, spherical harmonics, wavelets, ...
- Defined at discrete points

Vector Quantization: Example

Linear Transform Coding

Coefficients: $\Theta = \sum_i \phi_i (x_j) = \sum_i x_j a_{ij}$
- $\Theta_i = i^{th}$ resulting coefficient
- $x_j = j^{th}$ input value
- $a_{ij} = ij^{th}$ transform coefficient = $\phi_i (j)$

In matrix notation:
$\Theta = Ax$
$\Theta = A^{-1} \Theta$

Where $A$ is an $n \times n$ matrix, and each row
defines a basis function
### Example: Cosine Transform

- \( \phi_n(j) \)
- \( \phi_\ell(j) \)
- \( \phi_d(j) \)

\[ \Theta_i = \sum_j x_j \phi_i(j) \]

### Other Transforms

**Polynomial:**

- \( 1 \)
- \( x \)
- \( x^2 \)

**Wavelet (Haar):**

### How to Pick a Transform

**Goals:**
- Decorrelate
- Low coefficients for many terms
- Basis functions that can be ignored by perception

Why is using a Cosine of Fourier transform across a whole image bad?

How might we fix this?

### Usefulness of Transform

Typically transforms \( A \) are orthonormal: \( A^{-1} = A^T \)

**Properties of orthonormal transforms:**

\[ \sum x^2 = \sum \Theta^2 \] (energy conservation)

Would like to compact energy into as few coefficients as possible

\[ G_{TC} = \frac{1}{n} \sum \sigma_i^2 \left( \frac{1}{\prod \sigma_i} \right)^{1/n} \] (the transform coding gain)

\[ \sigma_i = \Theta_i - \Theta_m \]

The higher the gain, the better the compression
Case Study: JPEG

A nice example since it uses many techniques:
- Transform coding (Cosine transform)
- Scalar quantization
- Difference coding
- Run-length coding
- Huffman or arithmetic coding

JPEG (Joint Photographic Experts Group) was designed in 1991 for lossy and lossless compression of color or grayscale images. The lossless version is rarely used. Can be adjusted for compression ratio (typically 10:1)

JPEG: Quantization Table

| 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 |

Also divided through uniformity by a quality factor which is under control.

JPEG: Block scanning order

Uses run-length coding for sequences of zeros
**JPEG: example**

.125 bits/pixel (factor of 200)

---

**Case Study: MPEG**

Pretty much JPEG with *interframe coding*

Three types of frames

- I = intra frame (approx. JPEG) anchors
- P = predictive coded frames
- B = bidirectionally predictive coded frames

**Example:**

- **Type:** I B B P B B P B P B B I
- **Order:** 1 3 4 2 6 7 5 9 10 8 12 13 11

I frames are used for random access.

---

**MPEG matching between frames**

---

**MPEG: Compression Ratio**

356 x 240 image

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>18KB</td>
<td>7/1</td>
</tr>
<tr>
<td>P</td>
<td>6KB</td>
<td>20/1</td>
</tr>
<tr>
<td>B</td>
<td>2.5KB</td>
<td>50/1</td>
</tr>
<tr>
<td>Average</td>
<td>4.8KB</td>
<td>27/1</td>
</tr>
</tbody>
</table>

30 frames/sec x 4.8KB/frame x 8 bits/byte = 1.2 Mbits/sec + .25 Mbits/sec (stereo audio)

HDTV has 15x more pixels = 18 Mbits/sec
**MPEG in the “real world”**

- DVDs
  - Adds “encryption” and error correcting codes
- Direct broadcast satellite
- HDTV standard
  - Adds error correcting code on top
- Storage Tech “Media Vault”
  - Stores 25,000 movies

Encoding is much more expensive than encoding. Still requires special purpose hardware for high resolution and good compression.

---

**Wavelet Compression**

- A set of localized basis functions
- Avoids the need to block

"mother function" \( \phi(x) \)

\[
\phi_{il}(x) = \phi(2^{s}x - l)
\]

\( s = \text{scale} \quad l = \text{location} \)

**Requirements**

\[
\int_{-\infty}^{\infty} \psi(x) dx = 0 \quad \text{and} \quad \int_{-\infty}^{\infty} |\psi(x)|^2 dx < -\infty
\]

Many mother functions have been suggested.

---

**Haar Wavelets**

Most described, least used.

<table>
<thead>
<tr>
<th>( \phi(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 \leq x &lt; 1/2</td>
</tr>
<tr>
<td>-1 1/2 \leq x &lt; 1</td>
</tr>
<tr>
<td>0 otherwise</td>
</tr>
</tbody>
</table>

\( \psi_{il}(x) = \phi(2^{s}x - l) \)

---

**Haar Wavelet in 2d**

+ DC component = \( 2^{k+1} \) components
Discrete Haar Wavelet Transform

How do we convert this to the wavelet coefficients?

```c
for (j = n/2; j >= 1; j = j/2) {
    for (i = 1; i < j; i++) {
        b[i] = (a[2i-1] + a[2i])/2;
        b[j+i] = (a[2i-1] - a[2i])/2;
    }
    a[1..j] = b[1..j]; }
```

Linear time!

Haar Wavelet Transform: example

```plaintext
a = 2 1 2 -1 -2 0 2 -2
    = 1.5 .5 -1 0 .5 1.5 -1 2
= 1 -.5 .5 -.5
b = .25 .75 .5 .5 .5 1.5 -1 2
```

Wavelet decomposition

Morlet Wavelet

\[ \phi(x) = \text{Gaussian} \cdot \text{Cosine} = e^{-\frac{x^2}{2}} \cos(5x) \]

Corresponds to wavepackets in physics.

Heisenberg's uncertainty principle: a particle cannot be localized in space and momentum (frequency)
**JPEG2000**

**Overall Goals:**
- High compression efficiency with good quality at compression ratios of .25bpp
- Handle large images (up to $2^{32} \times 2^{32}$)
- Progressive image transmission
  - Quality, resolution or region of interest
- Fast access to various points in compressed stream
- Pan and Zoom while only decompressing parts
- Error resilience

---

**JPEG2000: Outline**

Main similarities with JPEG
- Separates into Y, I, Q color planes, and can downsample the I and Q planes
- Transform coding

Main differences with JPEG
- Wavelet transform
  - Daubechies 9-tap/7-tap (irreversible)
  - Daubechies 5-tap/3-tap (reversible)
- Many levels of hierarchy (resolution and spatial)
- Only arithmetic coding

---

**JPEG2000: 5-tap/3-tap**

$h[i] = a[2i-1] + (a[2i] + a[2i-2])/2$;

$l[i] = a[2i] + (h[i-1] + h[i] + 2)/2$;

$h[i]$: is the "high pass" filter, ie, the differences it depends on 3 values from a (3-tap)

$l[i]$: is the "low pass" filter, ie, the averages it depends on 5 values from a (5-tap)

Need to deal with boundary effects.
This is reversible: assignment
**JPEG 2000: Outline**

A spatial and resolution hierarchy
- **Tiles**: Makes it easy to decode sections of an image. For our purposes we can imagine the whole image as one tile.
- **Resolution Levels**: These are based on the wavelet transform. High-detail vs. Low detail.
- **Precinct Partitions**: Used within each resolution level to represent a region of space.
- **Code Blocks**: blocks within a precinct
- **Bit Planes**: ordering of significance of the bits

---

**JPEG vs. JPEG2000**

JPEG: .125bpp  
JPEG2000: .125bpp