Compression

- What
  - Reduce the amount of information (bits) needed to represent image
  - Video: 720 x 480 res, 30 fps, color
    - 720x480x20x3 = 31,104,000 bytes/sec
    - 30x60x120 = 216 Gigabytes for a 2 hour movie
- Why
  - Transmission (send video over wireless channel)
  - Storage (fit a 2 hour movie on a DVD)

Compression Model

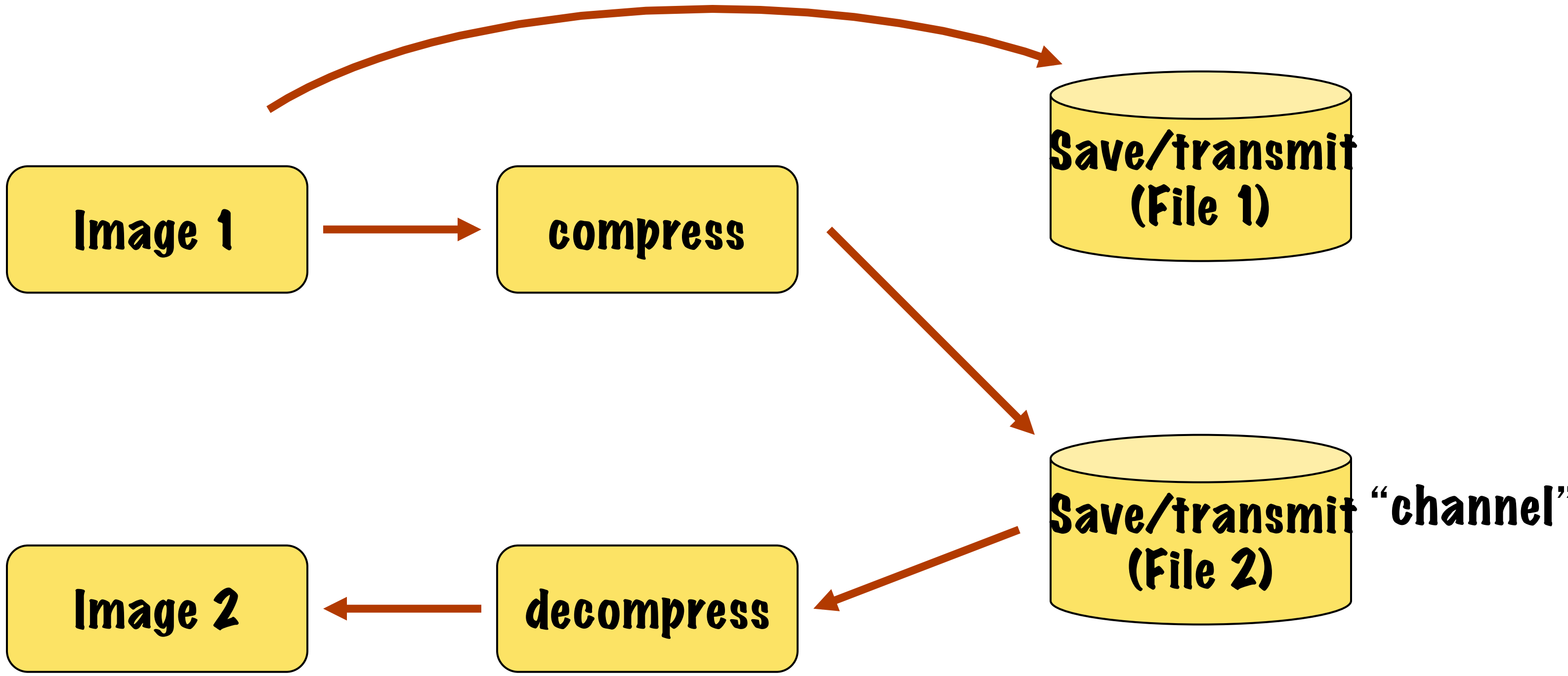


Image1 == Image2 -> "lossless" <- reduces redundant info

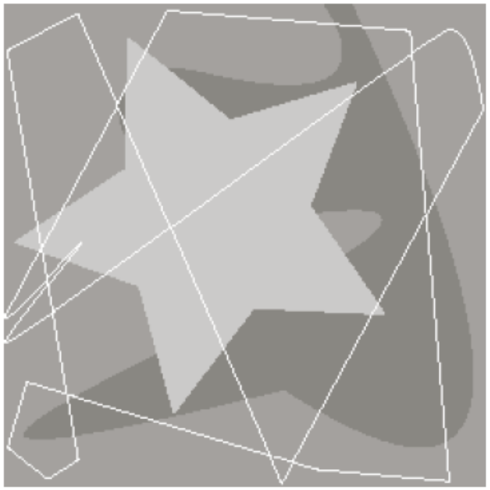
Image1 != Image2 -> "lossy" <- tries to reduce redundant & irrelevant info

Size(File1)/Size(File2) -> "compression ratio"

Redundancy

- Coding redundancy
  - More bits than necessary to create unique codes
- Spatiotemporal redundancy
  - Correlation between pixels
  - Patterns in image, motion in video
- Irrelevant information
  - Human visual system cannot distinguish more than a certain number of gray levels in a given image

1. Coding redundancy



$$p_r(r_k) = \frac{n_k}{MN}$$

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

$l(r_k)$ Number of bits needed for level k

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

Code 1: L_{avg} = 8 bits

Code 2: L_{avg} = 0.25*2+047*1+0.25*3+0.03*3=1.81 bits

Compression ratio C = 8/1.81 = 4.42

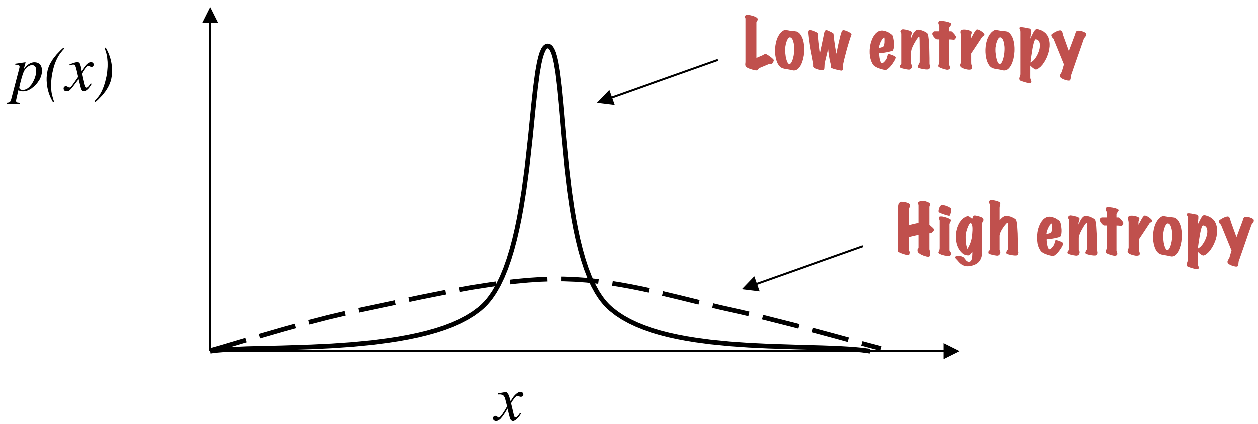
Information Theory

- How much information does a random event E give us?
  - I(E)=log[1/P(E)] = -log P(E)
  - Base 2 log, unit of information is **bits**
  - If an event is sure to happen, i.e. P(E)=1, do I get much information when it happens?
    - *Event: The sun rose this morning. Info: The world didn't end?*
  - If an event is unlikely to happen, small P(E), if it happens we get a lot of information.
    - *Event: It is raining in my bedroom. Info: my roof must be leaking.*

Information Theory

- Information content of a signal -> entropy

$$H = -\sum_{j=0}^{L-1} P(r_j)\log\left(P(r_j)\right) \qquad H = -\int p(x)\log p(x)\,dx$$



- Entropy gives the lower bound on #bits need to unambiguously represent a sequence of symbols

Back to coding example

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

Code 1: L_{avg} = 8 bits / pixel
Code 2: L_{avg} = 0.25*2+047*1+0.25*3+0.03*3=1.81 bits /pixel
Compression ratio C = 8/1.81 = 4.42

What does information theory say? (all logs are base 2)

H=-[0.25 log 0.25 + 0.47 log 0.47 + 0.25 log 0.25 + 0.03 log 0.03]

H=1.6614 bits / pixel. This is a lower bound. Could be achieved theoretically if coding groups of symbols rather than one at a time

Huffman Coding

- Need a method for computing the code
- Optimal when coding one symbol at a time
  - Variable length code
- Source reduction step
  - Order probabilities
  - Combine lowest probability pair into compound symbol, repeat.

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Huffman Coding

- Code assignment step
  - Start at the end, assign 0 and 1 to the two compound symbols, work backwards

Original source			Source reduction			
Symbol	Probability	Code	1	2	3	4
a_2	0.4	1	0.4 1	0.4 1	0.4 1	0.6 0
a_6	0.3	00	0.3 00	0.3 00	0.3 00	0.4 1
a_1	0.1	011	0.1 011	0.2 010	0.3 01	
a_4	0.1	0100	0.1 0100	0.1 011		
a_3	0.06	01010	0.1 0101			
a_5	0.04	01011				

L_{avg} = 0.4*1 + 0.3*2 + 0.1*3 + 0.1*4 + 0.06 *5 + 0.04*5 = 2.2 bits/pixel

H=[0.4log0.4+0.3log0.3+0.1log0.1+0.1*log0.01+0.06log0.06+0.04log0.04]= 2.1435 bits/pixel

JPEG, MPEG,H.261-H.264

Huffman coding

- After the code is created, it serves as a look-up table for coding and lossless decoding
- Block code: each source symbol is mapped into a fixed sequence of code symbols
- Instantaneous: Each code work can be decoded without reference to previous symbols in the sequence.
- Unique: Any sequence of symbols can be decoded in only one way.

Coding example

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

- Generate the code for the sequence $a_3\ a_1\ a_2\ a_2\ a_6$
  - a_3 a_1 a_2 a_2 a_6
  - 01010 011 1 1 00
- 010100111100

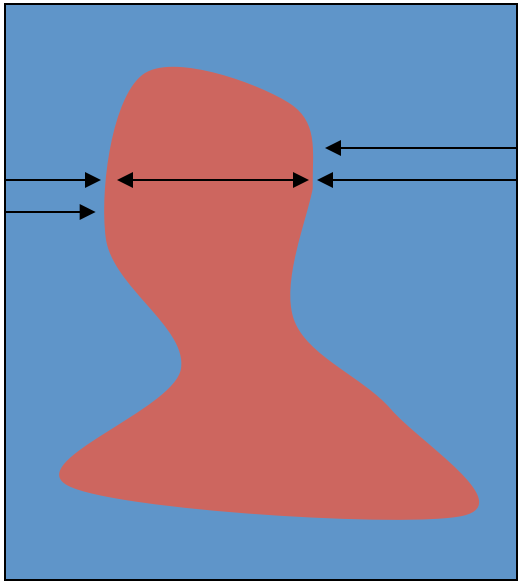
Decoding example

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

- Decode 010100111100.
  - Find first valid code word 010100111100 -> a_3
  - Find next valid code word 010100111100 -> $a_3 a_1$
  - 01010 011 1 1 00 -> a_3 a_1 a_2 a_2 a_6
- a_3 a_1 a_2 a_2 a_6

2. Spatial redundancy

- Images have homogeneous regions
- Run-length encoding
  - Row-major order
  - Encode value of “run” and it’s length
  - Can combine with symbol encoder
- Good for images with few, discrete color values. For instance, binary images
- Issues
  - How homogeneous is the data?
  - Is there enough continuity in rows?



BMP, JPEG,MPEG

BMP file format

- Uses a form of RLE. Two modes
  - Encoded: 2 byte representation: first byte number of pixels, second byte value/color index
  - Absolute: First byte 0. Second byte has special meaning

Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3–255	Specify pixels individually

Next 2 bytes position

This many uncompressed pixels follow

Fixed Length Codes

- Dictionary with strategy to capture special structure of data
- Example: LZW (Lempel-Ziv-Welch)
  - Start with basic dictionary (e.g. grey levels)
  - As new sequences of symbols are encountered add them to dictionary
    - Hope: encode frequently occuring sequences of symbols
      - Greedy
  - Can decompress w/out table

LZW coding

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

4 x 4 image
512-word dictionary
0-255 stores intensities
256-511 initially unused
Process left-to-right, top-to-bottom fashion

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

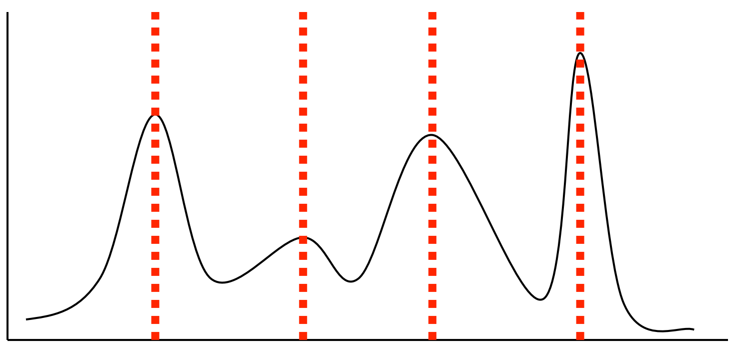
4 x 4 image
512-word dictionary
0-255 stores intensities
256-511 initially unused
Process left-to-right, top-to-bottom fashion

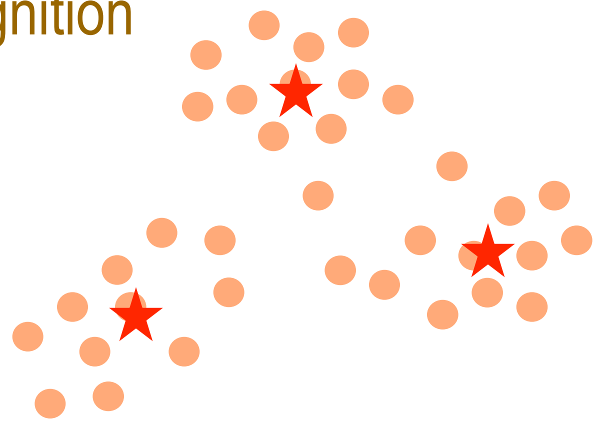
Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

GIF , TIFF , PDF

3. Irrelevant information/ Quantization

- Eliminate symbols that are too small or not important
- Find a small set of approximating symbols (less entropy)
  - Grey level or “vector quantization”
  - Find values that minimize error
  - Related to “clustering” in pattern recognition





- Quantization corresponds to a transformation $Q(f)$

4 levels

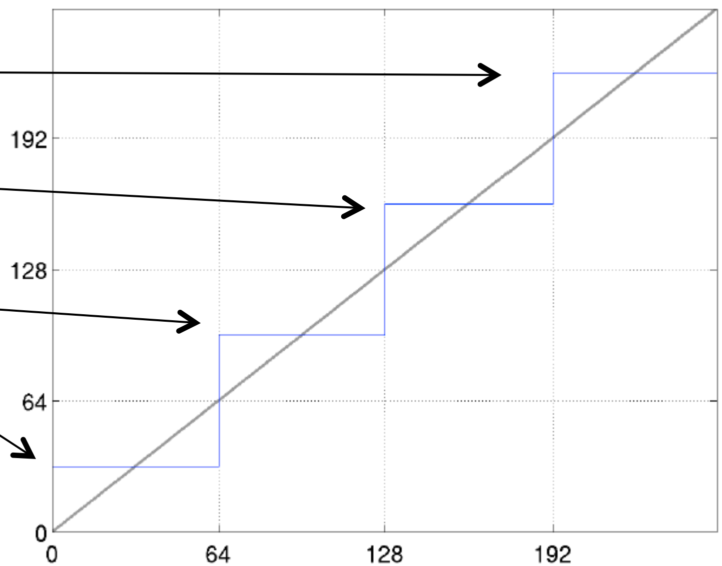
Code:

11

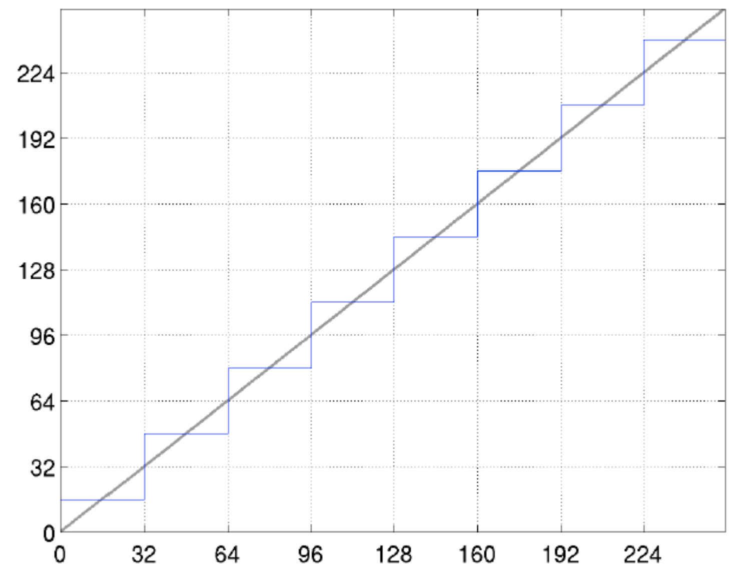
10

01

00



8 levels



Could also use Huffman coding to generate a more optimal code using the histogram of the image

Fidelity criteria

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]^2}$$

Approximation
For instance
quantized image

Input

Can we choose the quantization transform Q(f) to minimize SNR?

Lloyd-Max algorithm

- Bin boundaries $\min x = L_1 < L_2 < \ldots < L_{n+1} = \max x$
- Replacement values $p_1, p_2, \ldots, p_n$
- $q(x) = p_j$ when $L_j <= x < L_{j+1}$
- Choose L and p to minimize

$$E(L,p) = \sum_{i=1}^{m} \left(x_i - q(x_i)\right)^2 = \sum_{j=1}^{n} \sum_{x_i \in [L_j, L_{j+1})} \left(x_i - p_j\right)^2$$

Lloyd-Max algorithm

$$E(L,p)=\sum_{j=1}^{n}\sum_{x_i\in[L_j,L_{j+1})}\left(x_i-p_j\right)^2$$

- Derivative with respect to p

$$\frac{\partial E}{\partial p_j}=\sum_{x_i\in[L_j,L_{j+1})}2\left(x_i-p_j\right)=0$$

$$p_j=\frac{\displaystyle\sum_{x_i\in[L_j,L_{j+1})}x_i}{\#\left\{i\,\middle|\,x_i\in[L_j,L_{j+1})\right\}}$$

Update rule for p_j

Average intensity of pixels with intensities in this interval

Size of the set

Lloyd-Max algorithm

$$E(L,p)=\sum_{j=1}^{n}\sum_{x_i\in[L_j,L_{j+1})}\left(x_i-p_j\right)^2$$

- Update rule for L
  - $L_j=0.5*(p_{j-1}+p_j)$. Why?
  - If you decrease L_j, you will be assigning some x_i to p_j even though it is closer to p_{j-1}
  - If you increase L_j, you will be assigning some x_i to p_{j-1} even though it is closer to p_j

Lloyd-Max algorithm

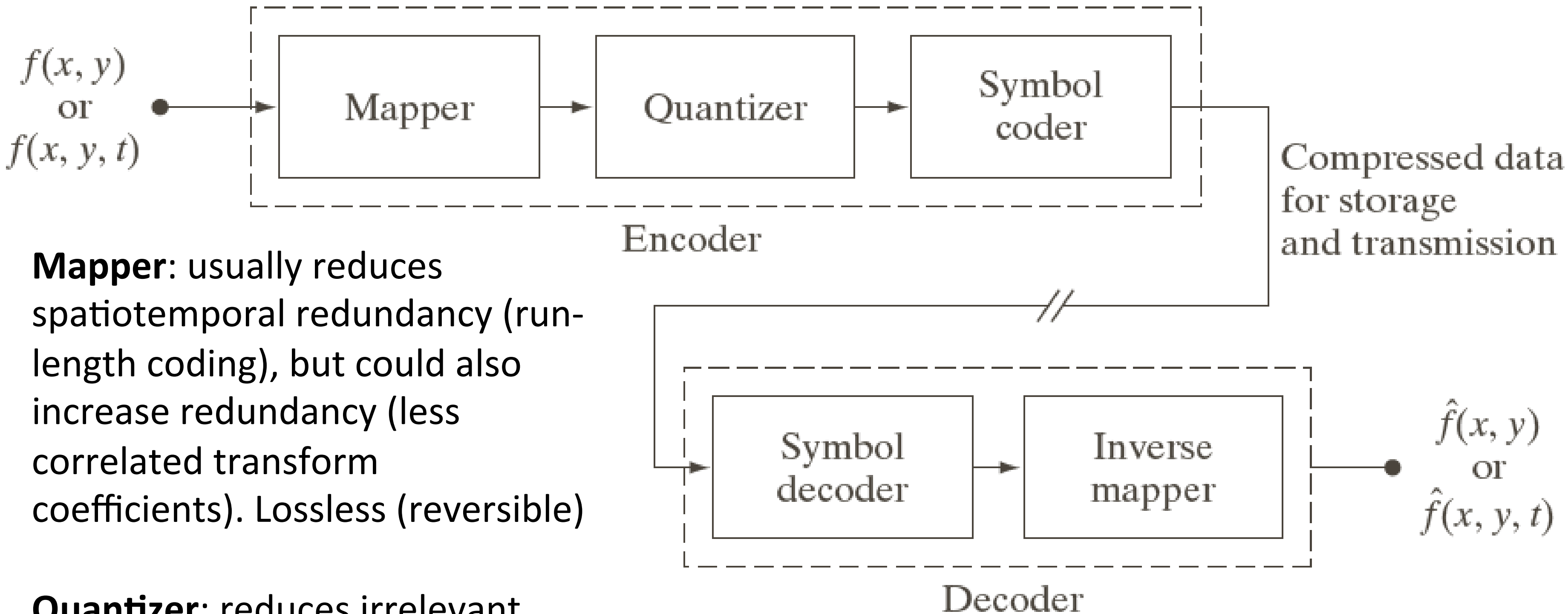
1. Start from uniform quantization bin boundaries
2. Update p using

   for j = 1 to n

$$p_j = \frac{\sum_{x_i \in [L_j, L_{j+1})} x_i}{\#\left\{ i \middle| x_i \in [L_j, L_{j+1}) \right\}}$$

3. Update L using $L_j = 0.5*(p_{j-1}+p_j)$ for j=2 to n. Notice lower and upper limit always set to min and max of image.
4. Go to step 2 until convergence

Image Compression Model



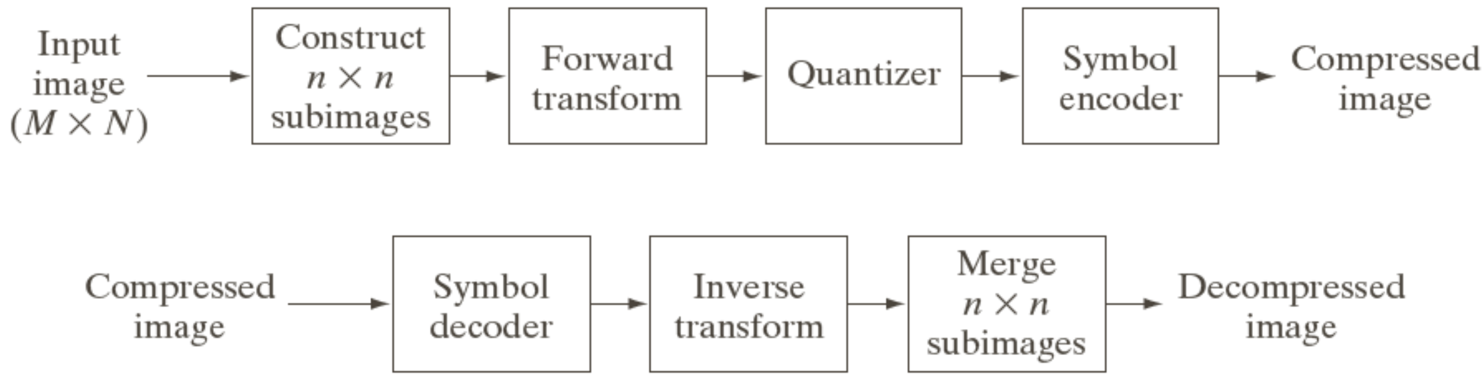
Mapper: usually reduces spatiotemporal redundancy (run-length coding), but could also increase redundancy (less correlated transform coefficients). Lossless (reversible)

Quantizer: reduces irrelevant information according to a fidelity criterion. Lossy

Symbol coder: For instance Huffman coding. Lossless.

Decoder: Undoes these in reverse order. Quantization can't be undone

Block transform coding



- **Forward transform**: decorrelate pixels + pack as much info as possible into fewest transform coefficients.
- **Quantization**: eliminate or coarsely quantize least informative transform coefficients
- **Symbol encoder**: Variable length coding for quantized coefficients
- All steps can be adaptive to subimages or fixed globally

Transforms

$$T(u,v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x,y) r(x,y,u,v)$$

Forward transform

$$g(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v) s(x,y,u,v)$$

Inverse transform

Basis functions

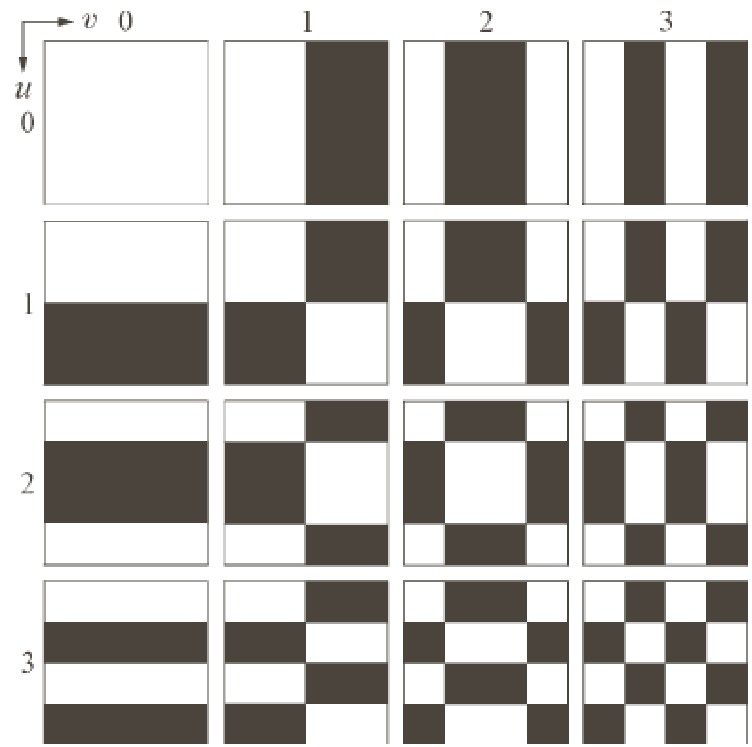
Fourier basis:

$r(x,y,u,v) = e^{-j2\pi(ux+vy)/n}$

$s(x,y,u,v) = (1/n^2)e^{j2\pi(ux+vy)/n}$

Walsh Hadamard transform (WHT)

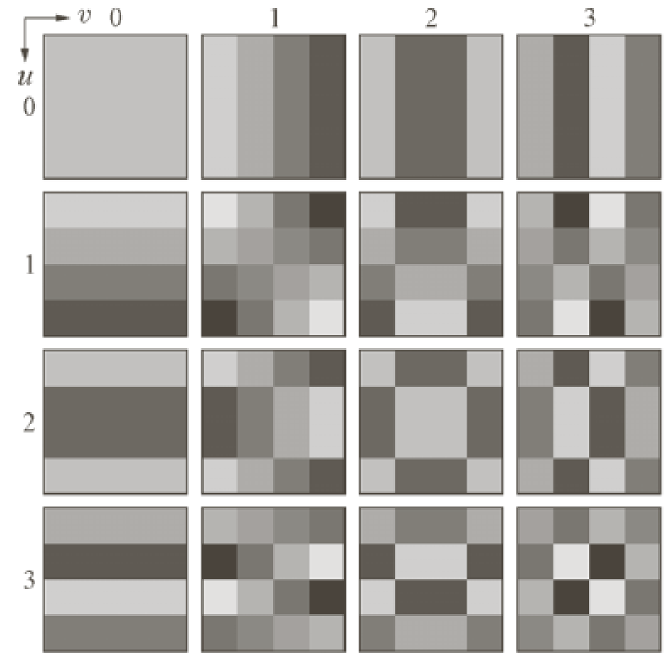
- A computationally very simple method. Basis functions consists of only +1 and -1
- Math for how they are produced in the textbook
- Notice frequency pattern
- Basis functions r and s are identical
- Example on the right is for n=4



Discrete Cosine Transform (DCT)

$$r(x,y,u,v) = s(x,y,u,v) = \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]$$

$$\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{n}} & u = 0 \\ \sqrt{\dfrac{2}{n}} & u = 1,2,\ldots,n-1 \end{cases}$$



Real valued basis functions
Used in JPEG

Fourier Walsh Hadamard Discrete Cosine

Reconstruction

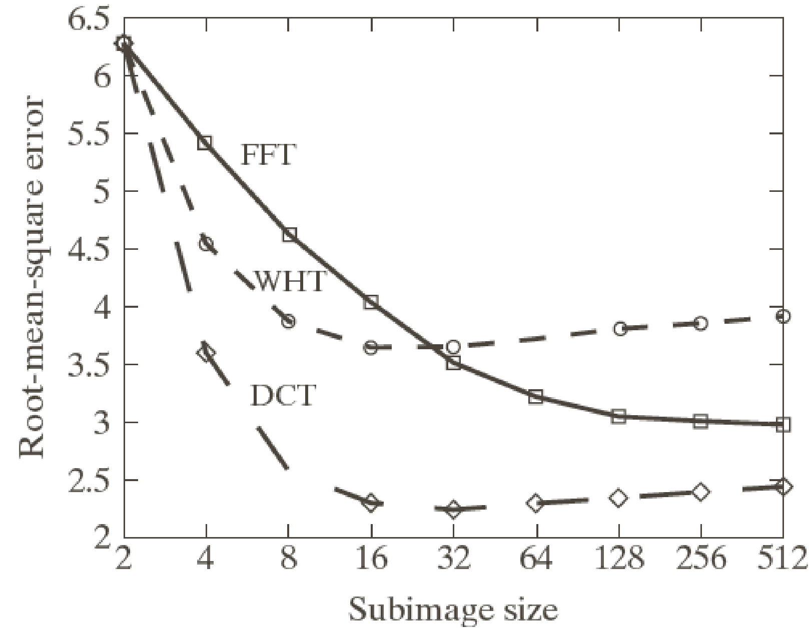
Error

8 x 8 blocks: 64 coefficients, in each block 32 coefficients with smallest magnitude were truncated. Blocks then reconstructed from remaining 32 coefficients.

RMS Errors: Fourier 2.32, WHT 1.78, DCT 1.13

Subimage size selection

- n is usually a power of 2
- Larger n increases computational complexity (transforms don't scale linearly with number of pixels in subimage)



25% coefficients retained

How to truncate

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	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

1	1	0	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	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

$$g(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v) X(u,v) s(x,y,u,v)$$

Top: zonal mask X(u,v). Fixed strategy
Bottom: threshold mask. Depends on subimage.
Have to also know which elements we are keeping

How to quantize

- Zonal strategy for how many bits used per coefficient for quantization

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Block Transform Coding: JPEG

- International standard (ISO)
- Baseline algorithm with extensions
- Transform: discrete cosine transform (DCT)
  - Encodes freq. Info w/out complex #s
  - FT of larger, mirrored signal
  - Does not have other nice prop. of FT

$$F_u = \alpha(u) \sum_{i=0}^{N-1} f_i \cos\left[\frac{(2i+1)u\pi}{2N}\right]$$

$$F_i = \sum_{u=0}^{N-1} \alpha(u) F_u \cos\left[\frac{(2i+1)u\pi}{2N}\right]$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0 \\ \sqrt{\frac{2}{N}} & u \neq 0 \end{cases}$$

JPEG Algorithm

- Integer grey-level image broken into 8x8 sub blocks
- Set middle (mean?) grey level to zero (subtract middle)
- DCT of sub blocks (11 bit precision) -> T(u,v)
- Rescale frequency components by Z(u,v) and round

Rescaling

$$\hat{T}(u,v) = \mathrm{round}\left(\frac{\mathrm{T(u,v)}}{\mathrm{Z(u,v)}}\right)$$

- Different scalling matrices possible, but recommended is:

$$Z(u,v) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Reordering

- DCT entries reordered in zig-zag fashion to increase coherency (produce blocks of zeros)

$$\begin{bmatrix} 0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{bmatrix}$$

Coding

- Each sub-block is coded as a difference from previous sub-block
- Zeros are run-length encoded and nonzero elements are Huffman coded
  - Modified HC to allow for zeros

JPEG Example
Compression Ratio ~10:1