COM303: Digital Signal Processing

Lecture 21: Image Compression

overview

- introduction to quantization
- the problem of image compression
- the JPEG standard

quantization: the basics

Quantization

- digital devices can only deal with integers (R bits per sample)
- we need to map the range of a signal onto a finite set of values
- irreversible loss of information $\rightarrow$ quantization noise

Quantization

- digital devices can only deal with integers (R bits per sample)
- we need to map the range of a signal onto a finite set of values
- irreversible loss of information $\rightarrow$ quantization noise

Quantization

- digital devices can only deal with integers (R bits per sample)
- we need to map the range of a signal onto a finite set of values
- irreversible loss of information $\rightarrow$ quantization noise

Quantization strategies

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

Several factors at play:

- storage budget (R bits per sample)
- storage scheme (fixed point, floating point)
- properties of the input
  - range
  - probability distribution

Quantization strategies

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

Several factors at play:

- storage budget (R bits per sample)
- storage scheme (fixed point, floating point)
- properties of the input
  - range
  - probability distribution

Quantization strategies

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

Several factors at play:

- storage budget (R bits per sample)
- storage scheme (fixed point, floating point)
- properties of the input
  - range
  - probability distribution

Quantization strategies

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

Several factors at play:

- storage budget (R bits per sample)
- storage scheme (fixed point, floating point)
- properties of the input
  - range
  - probability distribution

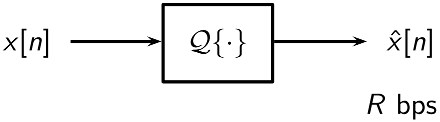
Quantization strategies

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

Several factors at play:

- storage budget (R bits per sample)
- storage scheme (fixed point, floating point)
- properties of the input
  - range
  - probability distribution

Uniform scalar quantization



The simplest quantizer:

- each sample is encoded individually (hence *scalar*) using R bits
- each sample is quantized independently (memoryless quantization)
- the input range is divided into 2^R equal-size intervals

Uniform scalar quantization

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

R bps

The simplest quantizer:

- each sample is encoded individually (hence *scalar*) using R bits
- each sample is quantized independently (memoryless quantization)
- the input range is divided into 2^R equal-size intervals

Uniform scalar quantization

$$x[n] \longrightarrow \boxed{\mathcal{Q}\{\cdot\}} \longrightarrow \hat{x}[n]$$

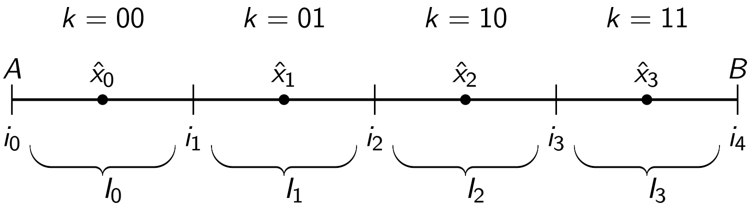
R bps

The simplest quantizer:

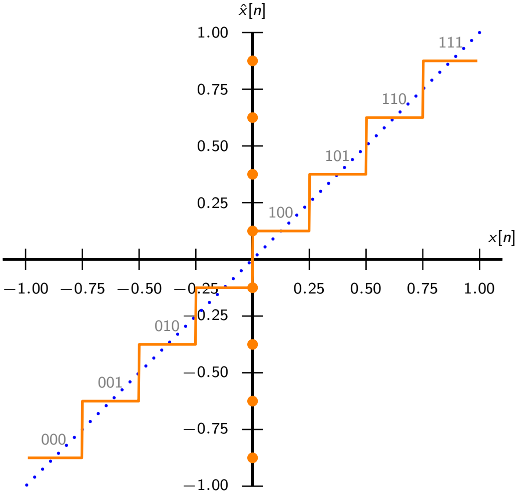
- each sample is encoded individually (hence *scalar*) using R bits
- each sample is quantized independently (memoryless quantization)
- the input range is divided into 2^R equal-size intervals

Uniform scalar quantization

- input assumed uniformly distributed over $[A, B]$
- range is split into 2^R *equal* intervals of width $\Delta = (B - A)2^{-R}$
- quantized value is interval's midpoint



Uniform 3-Bit quantization function



Quantization Error

$$e[n] = \mathcal{Q}\{x[n]\} - x[n] = \hat{x}[n] - x[n]$$

- model $x[n]$ as a stochastic process
- model error as a white noise sequence:
  - error samples are uncorrelated
  - all error samples have the same distribution

Quantization Error

$$e[n] = \mathcal{Q}\{x[n]\} - x[n] = \hat{x}[n] - x[n]$$

- model $x[n]$ as a stochastic process
- model error as a white noise sequence:
  - error samples are uncorrelated
  - all error samples have the same distribution

Quantization Error

$$e[n] = \mathcal{Q}\{x[n]\} - x[n] = \hat{x}[n] - x[n]$$

- model $x[n]$ as a stochastic process
- model error as a white noise sequence:
  - error samples are uncorrelated
  - all error samples have the same distribution

Quantization Error

$$e[n] = \mathcal{Q}\{x[n]\} - x[n] = \hat{x}[n] - x[n]$$

- model $x[n]$ as a stochastic process
- model error as a white noise sequence:
  - error samples are uncorrelated
  - all error samples have the same distribution

Error analysis for uniform iid input

- error energy

$$\sigma_e^2 = \Delta^2/12, \qquad \Delta = (B - A)/2^R$$

- signal energy

$$\sigma_x^2 = (B - A)^2/12$$

- signal to noise ratio

$$\mathrm{SNR} = 2^{2R}$$

- in dB

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} 2^{2R} \approx 6R \text{ dB}$$

Error analysis for uniform iid input

- error energy

$$\sigma_e^2 = \Delta^2/12, \qquad \Delta = (B - A)/2^R$$

- signal energy

$$\sigma_x^2 = (B - A)^2/12$$

- signal to noise ratio

$$\mathrm{SNR} = 2^{2R}$$

- in dB

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} 2^{2R} \approx 6R \text{ dB}$$

Error analysis for uniform iid input

- error energy

$$\sigma_e^2 = \Delta^2/12, \qquad \Delta = (B-A)/2^R$$

- signal energy

$$\sigma_x^2 = (B-A)^2/12$$

- signal to noise ratio

$$\mathrm{SNR} = 2^{2R}$$

- in dB

$$\mathrm{SNR}_{\mathrm{dB}} = 10\log_{10} 2^{2R} \approx 6R \text{ dB}$$

Error analysis for uniform iid input

- error energy

$$\sigma_e^2 = \Delta^2/12, \qquad \Delta = (B-A)/2^R$$

- signal energy

$$\sigma_x^2 = (B-A)^2/12$$

- signal to noise ratio

$$\mathrm{SNR} = 2^{2R}$$

- in dB

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} 2^{2R} \approx 6R \text{ dB}$$

The "6dB/bit" rule of thumb

- a compact disk has 16 bits/sample:

$$\text{max SNR} = 96\text{dB}$$

- a DVD has 24 bits/sample:

$$\text{max SNR} = 144\text{dB}$$

The "6dB/bit" rule of thumb

- a compact disk has 16 bits/sample:

$$\max \text{SNR} = 96\text{dB}$$

- a DVD has 24 bits/sample:

$$\max \text{SNR} = 144\text{dB}$$

Rate/Distortion Curve

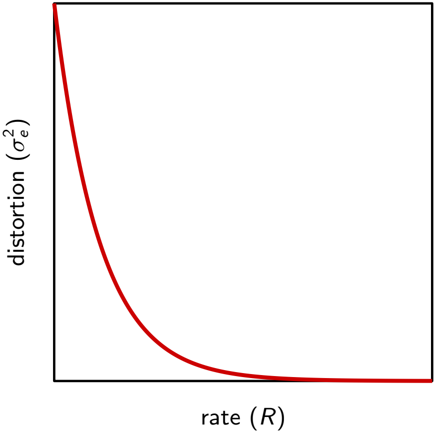


image compression fundamentals

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits

- total number of possible images: $2^{524{,}288} \approx 10^{157{,}826}$
- number of atoms in the universe: 10^{82}

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits
- total number of possible images: $2^{524,288} \approx 10^{157,826}$
- number of atoms in the universe: 10^{82}

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits
- total number of possible images: $2^{524,288} \approx 10^{157,826}$
- number of atoms in the universe: 10^{82}

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits
- total number of possible images: $2^{524,288} \approx 10^{157,826}$
- number of atoms in the universe: 10^{82}

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number
- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an “encyclopedia of images”
- to indicate an image, simply give its number
- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number
- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number

- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number
- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number

- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

Compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

- exploit the local spatial correlation
- compress remote regions independently

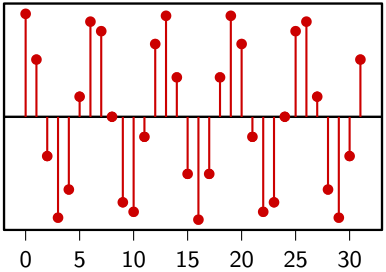
Compressing at block level

- exploit the local spatial correlation
- compress remote regions independently

Transform coding

A simple example:

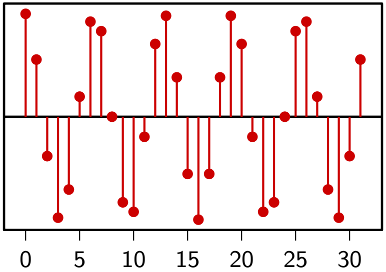
- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



Transform coding

A simple example:

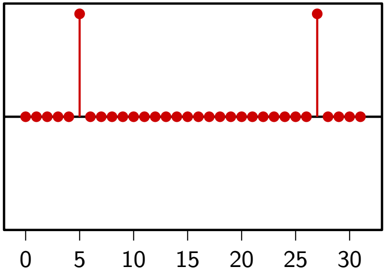
- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



Transform coding

A simple example:

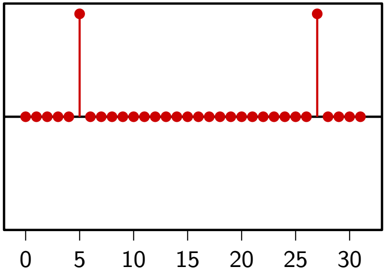
- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



Transform coding

A simple example:

- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



Transform coding

Ideally, we would like a transform that:

- captures the important features of an image block in a few coefficients
- is efficient to compute
- answer: the Discrete Cosine Transform

Transform coding

Ideally, we would like a transform that:

- captures the important features of an image block in a few coefficients
- is efficient to compute

- answer: the Discrete Cosine Transform

Transform coding

Ideally, we would like a transform that:

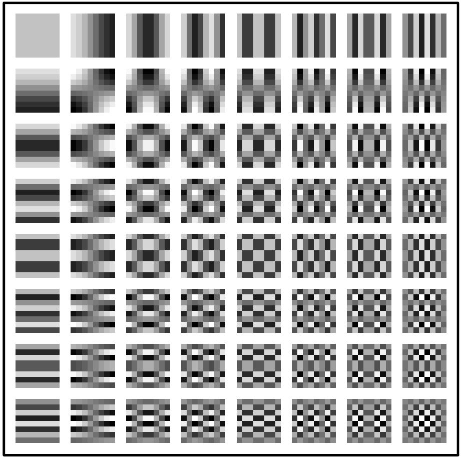
- captures the important features of an image block in a few coefficients
- is efficient to compute

- answer: the Discrete Cosine Transform

2D-DCT

$$C[k_1, k_2] = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x[n_1, n_2] \cos\left[\frac{\pi}{N}\left(n_1 + \frac{1}{2}\right) k_1\right] \cos\left[\frac{\pi}{N}\left(n_2 + \frac{1}{2}\right) k_2\right]$$

DCT basis vectors for an 8×8 image



Smart quantization

- deadzone
- variable step (fine to coarse)

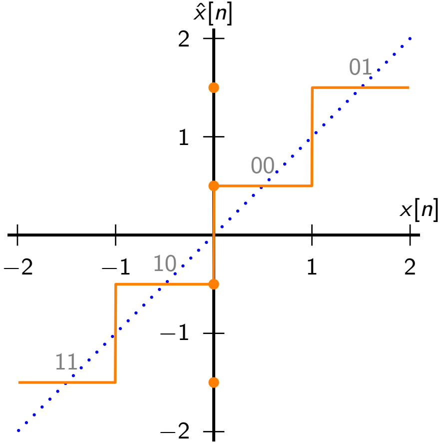
Smart quantization

- deadzone
- variable step (fine to coarse)

Quantization

Standard quantization:

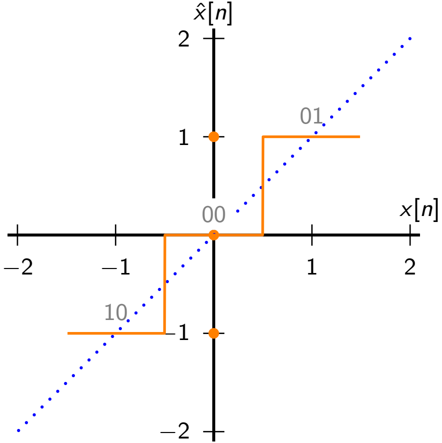
$$\hat{x} = \text{floor}(x) + 0.5$$



Quantization

Deadzone quantization:

$$\hat{x} = \text{round}(x)$$



Entropy coding

- minimize the effort to encode a certain amount of information
- associate short symbols to frequent values and vice-versa
- if it sounds familiar it's because it is...

Entropy coding

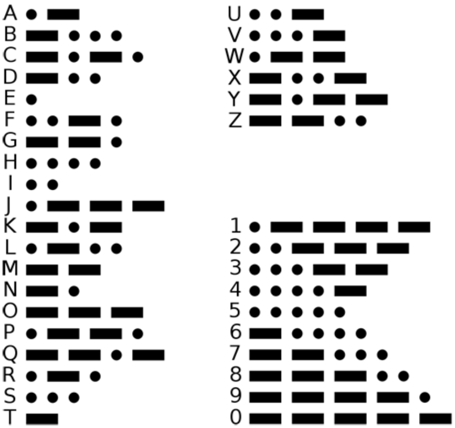
- minimize the effort to encode a certain amount of information
- associate short symbols to frequent values and vice-versa
- if it sounds familiar it's because it is...

Entropy coding

- minimize the effort to encode a certain amount of information
- associate short symbols to frequent values and vice-versa

- if it sounds familiar it's because it is...

Entropy coding



the JPEG standard

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- split image into 8×8 non-overlapping blocks
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- smart quantization
- entropy coding

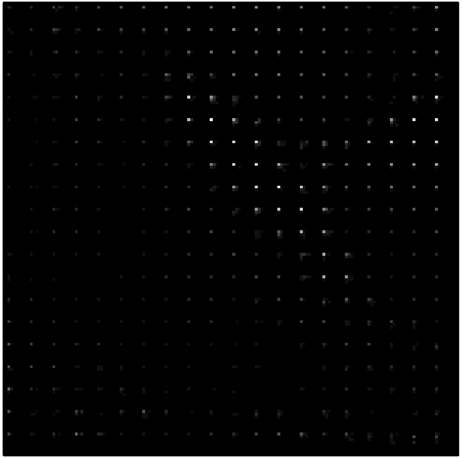
Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- quantize DCT coefficients according to psycovisually-tuned tables
- entropy coding

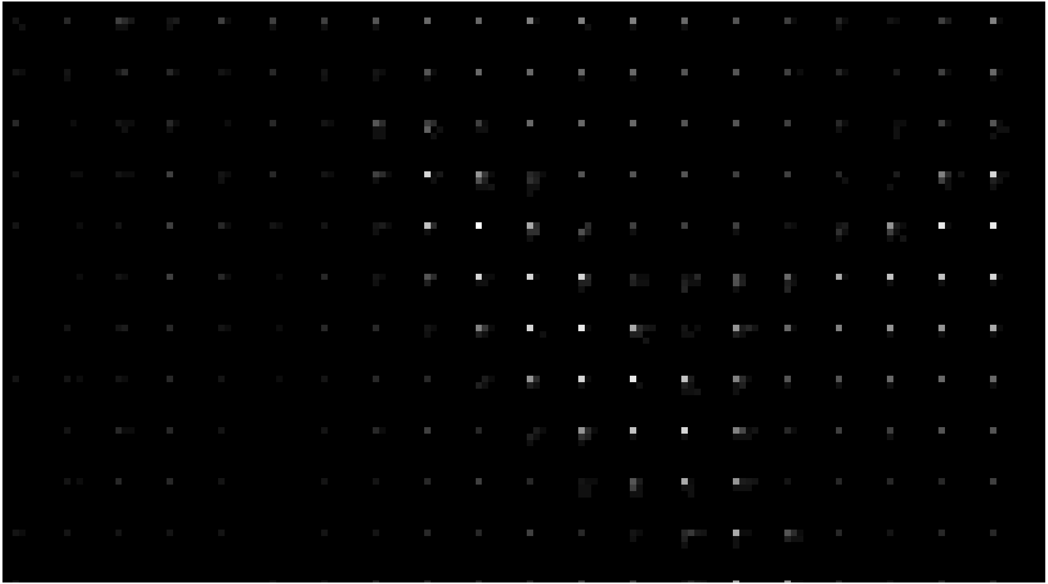
Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- quantize DCT coefficients according to psycovisually-tuned tables
- run-length encoding and Huffman coding

DCT coefficients of image blocks (detail)



DCT coefficients of image blocks (detail)



Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients have a higher visual impact
- find out the critical coefficients by experimentation
- use smaller quantization intervals (i.e. use more more bits) for the important coefficients

Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients have a higher visual impact
- find out the critical coefficients by experimentation
- use smaller quantization intervals (i.e. use more more bits) for the important coefficients

Smart quantization

- most coefficients are negligible → captured by the deadzone
- some coefficients have a higher visual impact
- find out the critical coefficients by experimentation
- use smaller quantization intervals (i.e. use more more bits) for the important coefficients

Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients have a higher visual impact
- find out the critical coefficients by experimentation
- use smaller quantization intervals (i.e. use more more bits) for the important coefficients

Psychovisually-tuned quantization table

$$\hat{c}[k_1, k_2] = \text{round}(c[k_1, k_2]/Q[k_1, k_2])$$

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

Advantages of nonuniform bit allocation at 0.2bpp

uniform

tuned

Advantages of nonuniform bit allocation (detail)

uniform

tuned

Efficient coding

- most coefficients are small, decreasing with index
- use zigzag scan to maximize ordering
- quantization will create long series of zeros

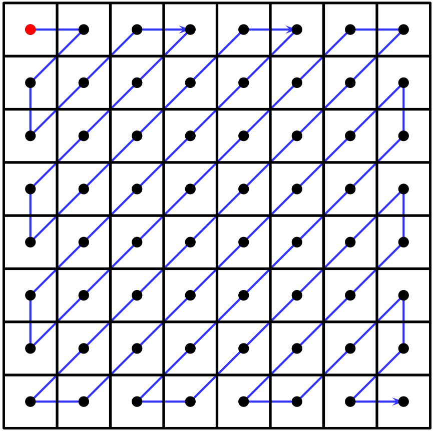
Efficient coding

- most coefficients are small, decreasing with index
- use zigzag scan to maximize ordering
- quantization will create long series of zeros

Efficient coding

- most coefficients are small, decreasing with index
- use zigzag scan to maximize ordering
- quantization will create long series of zeros

Zigzag scan



Example

$$
\begin{bmatrix}
100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

100, -60, 0, 0, 0, 0, 6, 0, 0, 0, 13, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

100, -60, 0, 0, 0, 0, 6, 0, 0, 0, 13, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Runlength encoding

- the DC value is encoded differentially wrt previous block
- each nonzero AC value is encoded as the triple

$$[(r, s),\ c]$$

  - r is the *runlength* i.e. the number of zeros before the current value
  - s is the *category* i.e. the number of bits needed to encode the value
  - c is the actual value
  - $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

- the DC value is encoded differentially wrt previous block
- each nonzero AC value is encoded as the triple

$$[(r, s),\ c]$$

  - r is the *runlength* i.e. the number of zeros before the current value
  - s is the *category* i.e. the number of bits needed to encode the value
  - c is the actual value
  - $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

- the DC value is encoded differentially wrt previous block
- each nonzero AC value is encoded as the triple

$$[(r, s),\ c]$$

  - r is the *runlength* i.e. the number of zeros before the current value
  - s is the *category* i.e. the number of bits needed to encode the value
  - c is the actual value
  - $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

- the DC value is encoded differentially wrt previous block
- each nonzero AC value is encoded as the triple

$$[(r, s),\ c]$$

  - r is the *runlength* i.e. the number of zeros before the current value
  - s is the *category* i.e. the number of bits needed to encode the value
  - c is the actual value
  - $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

- the DC value is encoded differentially wrt previous block
- each nonzero AC value is encoded as the triple

$$[(r, s),\ c]$$

  - r is the *runlength* i.e. the number of zeros before the current value
  - s is the *category* i.e. the number of bits needed to encode the value
  - c is the actual value
  - $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

- the DC value is encoded differentially wrt previous block
- each nonzero AC value is encoded as the triple

$$[(r, s),\ c]$$

  - r is the *runlength* i.e. the number of zeros before the current value
  - s is the *category* i.e. the number of bits needed to encode the value
  - c is the actual value
  - $(0, 0)$ indicates that from now on it's only zeros (end of block)

Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[100],\ [(0,6),\ -60],\ [(4,3),\ 6],\ [(3,4),\ 13],\ [(8,1),\ -1], [(0,0)]$$

Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[100],\ [(0,6),\ -60],\ [(4,3),\ 6],\ [(3,4),\ 13],\ [(8,1),\ -1], [(0,0)]$$

The runlength-size pairs

- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

The runlength-size pairs

- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

The runlength-size pairs

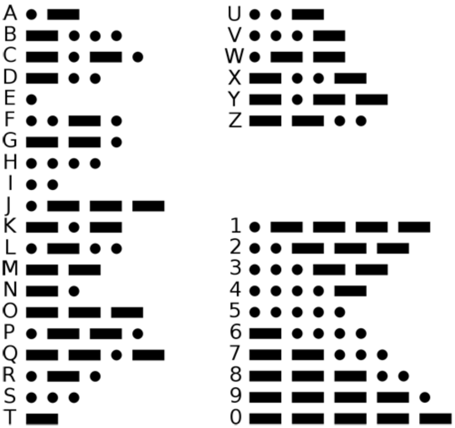
- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

The runlength-size pairs

- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

Variable-length encoding

great idea: shorter binary sequences for common symbols



Variable-length encoding

however: if symbols have different lengths, we must know how to parse them!

- in English, spaces separate words $\rightarrow$ extra symbol (wasteful)
- in Morse code, pauses separate letters and words (wasteful)
- can we do away with separators?

Variable-length encoding

however: if symbols have different lengths, we must know how to parse them!

- in English, spaces separate words $\rightarrow$ extra symbol (wasteful)
- in Morse code, pauses separate letters and words (wasteful)
- can we do away with separators?

Variable-length encoding

however: if symbols have different lengths, we must know how to parse them!

- in English, spaces separate words $\rightarrow$ extra symbol (wasteful)
- in Morse code, pauses separate letters and words (wasteful)
- can we do away with separators?

Prefix-free codes

- no valid sequence can be the beginning of another valid sequence
- can parse a bitstream sequentially with no look-ahead
- extremely easy to understand graphically...

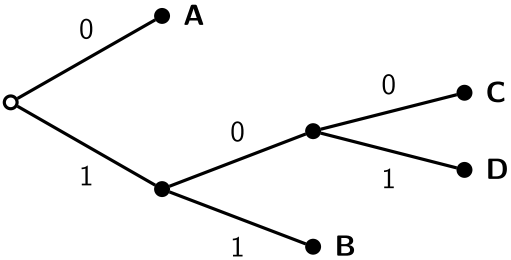
Prefix-free codes

- no valid sequence can be the beginning of another valid sequence
- can parse a bitstream sequentially with no look-ahead
- extremely easy to understand graphically...

Prefix-free codes

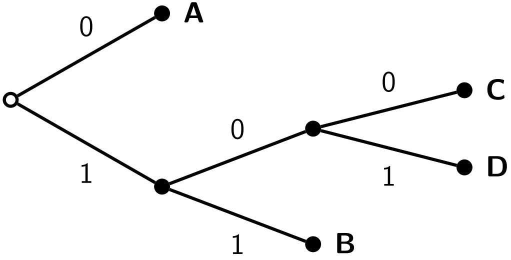
- no valid sequence can be the beginning of another valid sequence
- can parse a bitstream sequentially with no look-ahead
- extremely easy to understand graphically...

Prefix-free code



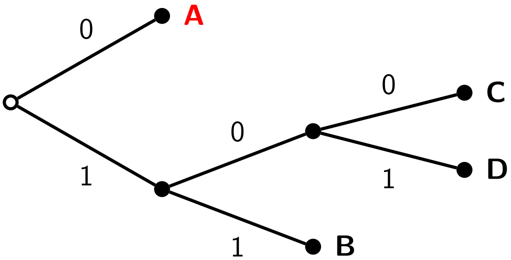
001100110101100

Prefix-free code



001100110101100

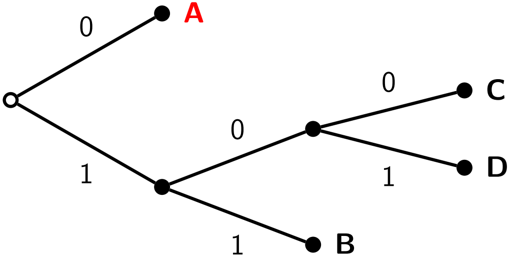
Prefix-free code



001100110101100

A

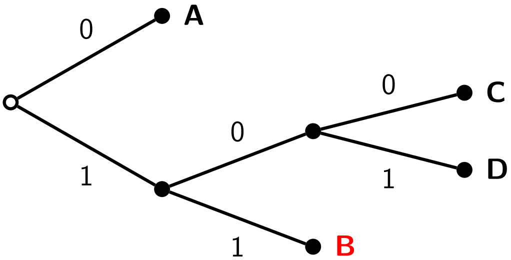
Prefix-free code



00110011010110 0

AA

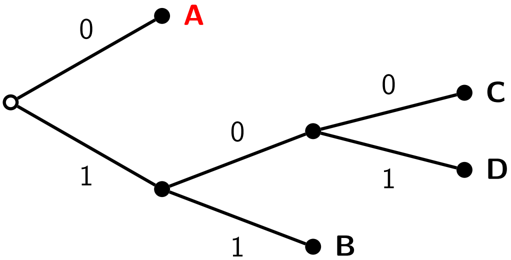
Prefix-free code



00110011010110 0

AAB

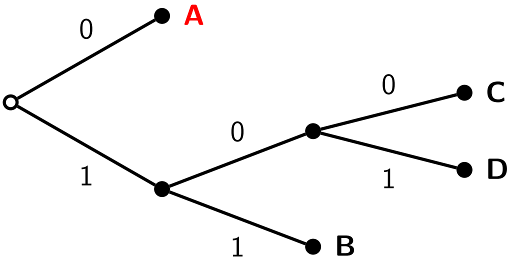
Prefix-free code



0011**0**0110101100

AABA

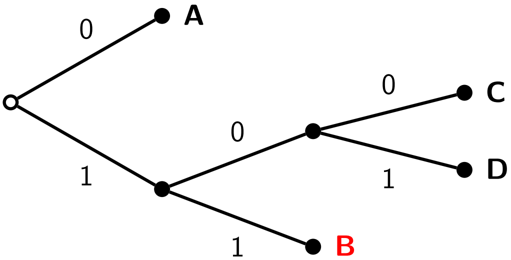
Prefix-free code



00110**0**110101100

AABAA

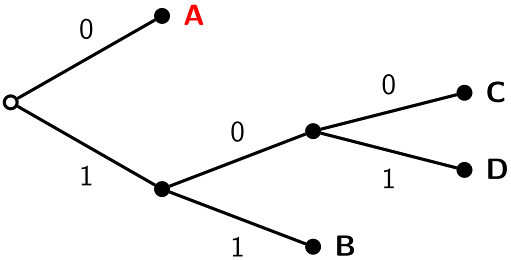
Prefix-free code



001100110101100

AABAAB

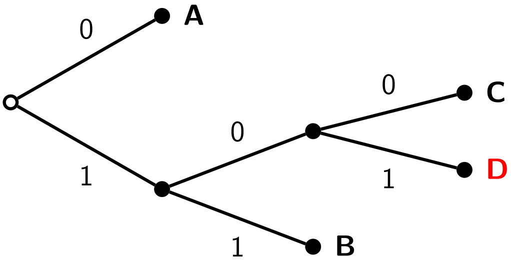
Prefix-free code



001100110101100

AABAABA

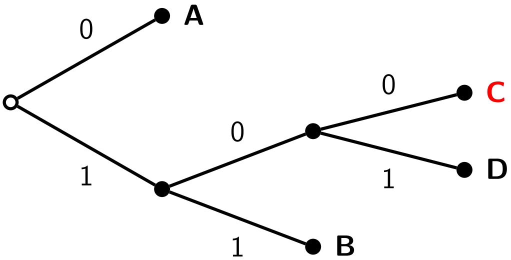
Prefix-free code



001100110101100

AABAABAD

Prefix-free code



00110011010110 0

AABAABADC

Entropy coding

goal: minimize message length

- assign short sequences to more frequent symbols
- the Huffman algorithm builds the optimal code for a set of symbol probabilities
- in JPEG, you can use a “general-purpose” Huffman code or build your own (but then you pay a “side-information” price)

Entropy coding

goal: minimize message length

- assign short sequences to more frequent symbols
- the Huffman algorithm builds the optimal code for a set of symbol probabilities
- in JPEG, you can use a "general-purpose" Huffman code or build your own (but then you pay a "side-information" price)

Entropy coding

goal: minimize message length

- assign short sequences to more frequent symbols
- the Huffman algorithm builds the optimal code for a set of symbol probabilities
- in JPEG, you can use a "general-purpose" Huffman code or build your own
  (but then you pay a "side-information" price)

Example

- four symbols: A, B, C, D
- probability table:

$$p(A) = 0.38 \qquad p(B) = 0.32$$

$$p(C) = 0.1 \qquad p(D) = 0.2$$

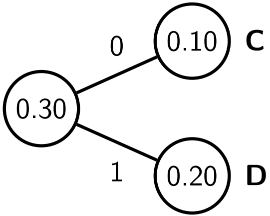
Example

- four symbols: A, B, C, D
- probability table:

$$p(A) = 0.38 \qquad p(B) = 0.32$$

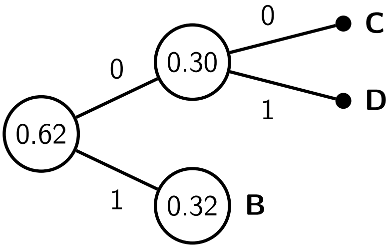
$$p(C) = 0.1 \qquad p(D) = 0.2$$

Building the Huffman code



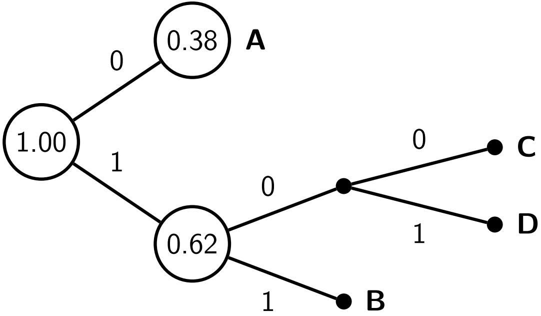
$p(A) = 0.38 \quad p(B) = 0.32 \quad p(C) = 0.1 \quad p(D) = 0.2$

Building the Huffman code



$p(A) = 0.38 \quad p(B) = 0.32 \quad p(C + D) = 0.3$

Huffman Coding



$p(A) = 0.38 \quad p(B + C + D) = 0.62$

Conclusions

- JPEG is a very complex and comprehensive standard:
  - lossless, lossy
  - color, B&W
  - progressive encoding
  - HDR (12bpp) for medical imaging
- JPEG is VERY good:
  - compression factor of 10:1 virtually indistinguishable
  - rates of 1bpp for RGB images acceptable (25:1 compression ratio)
- other important compression schemes:
  - TIFF, JPEG2000
  - MPEG (MP3)