Image Compression

CS 663, Ajit Rajwade

Image Compression

- Process of converting an image file into another image file that occupies less storage space, without sacrificing its *visual* content as far as possible.

- Useful for saving **storage space**, and **transmission costs**.

Types of compression

- **Lossless**: the compressed image can be converted back with zero error.

- **Lossy**: the compressed image cannot be converted back to the original without error. The amount of **error** is **inversely proportional** to the **storage space** (usually) and can be controlled by the user.

Lossless compression - examples

- LZW method (used in Winzip)
- Huffman encoding (part of the JPEG algorithm, although overall JPEG is lossy)
- Run-length encoding (also part of the JPEG algorithm, although JPEG is lossy overall)

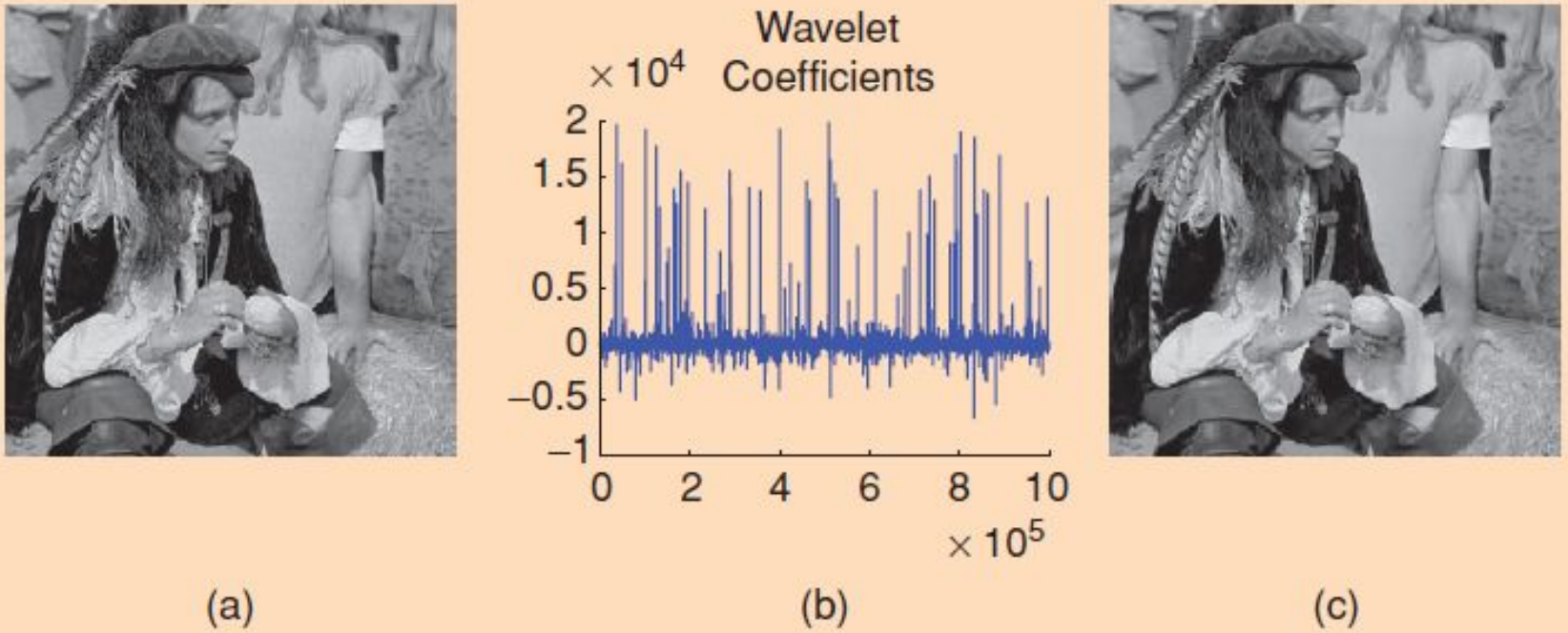
Lossy compression

- JPEG
- MPEG (for video)
- MP3 (for audio)
- Machine learning based techniques for compression of images or video (not covered in this course).

Lossy image compression

- Compression of text files or exe files cannot afford to be lossy.
- But some portion of image content is often not very noticeable to the human eye, especially the higher frequencies. Discarding this extraneous information leads to compression without significant loss of visual appeal.

Source: Article on compressive sensing by Candes and Wakin, from IEEE Signal Processing Magazine, 2008



[FIG1] (a) Original megapixel image with pixel values in the range [0,255] and (b) its wavelet transform coefficients (arranged in random order for enhanced visibility). Relatively few wavelet coefficients capture most of the signal energy; many such images are highly compressible. (c) The reconstruction obtained by zeroing out all the coefficients in the wavelet expansion but the 25,000 largest (pixel values are thresholded to the range [0,255]). The difference with the original picture is hardly noticeable.

JPEG compression method

- JPEG = Joint Photographic Experts Group
- One of the most popular standards for compression of photographic images – widely used on the internet.
- Widely used in digital cameras.
- Implemented in all standard image processing software (MATLAB, OpenCV, etc.)
- Essentially lossy (though there are some lossless variants)
- Applicable for color as well as grayscale images.

JPEG image compression

- User specifies a quality factor (*Q*) between 0 and 100 (higher *Q* means better quality)
- JPEG algorithm compresses the image based on the user-provided *Q*.
- Higher the *Q*, less will be the compression rate (but higher image quality). Lower *Q* will give higher compression rate (but poorer image quality).
- JPEG can achieve 1/10 or 1/15 compression rate with little loss of quality.

JPEG image compression

- How is the loss of quality measured?
- As MSE between original (uncompressed) and reconstructed images:

$$MSE(I_{\text{orig}}, I_{\text{compressed}}) = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(I_{\text{orig}}(i,j) - I_{\text{compressed}}(i,j)\right)^2$$

$$\text{Peak Signal to Noise Ratio} = PSNR = 10\log_{10}\left(\frac{255^2}{MSE}\right)$$

Q = 100, compression rate = 1/2.6

Q = 10, compression rate = 1/46

Q = 50, compression rate = 1/15

Q = 1, compression rate = 1/144

Q = 25, compression rate = 1/23

http://en.wikipedia.org/wiki/JPEG

Steps of the JPEG algorithm (encoder): Overview (approximate)

1. Divide the image into **non-overlapping 8 x 8 blocks** and compute the **discrete cosine transform** (DCT) of each block. This produces a set of 64 “DCT coefficients” per block.
2. Quantize these DCT coefficients, i.e. divide by some number and round off to nearest integer (that’s why it is lossy). Many coefficients now become 0 and need not be stored!
3. Now run a lossless compression algorithm (typically Huffman encoding) on the entire set of integers.

We will go through each step in detail in the several slides to follow.

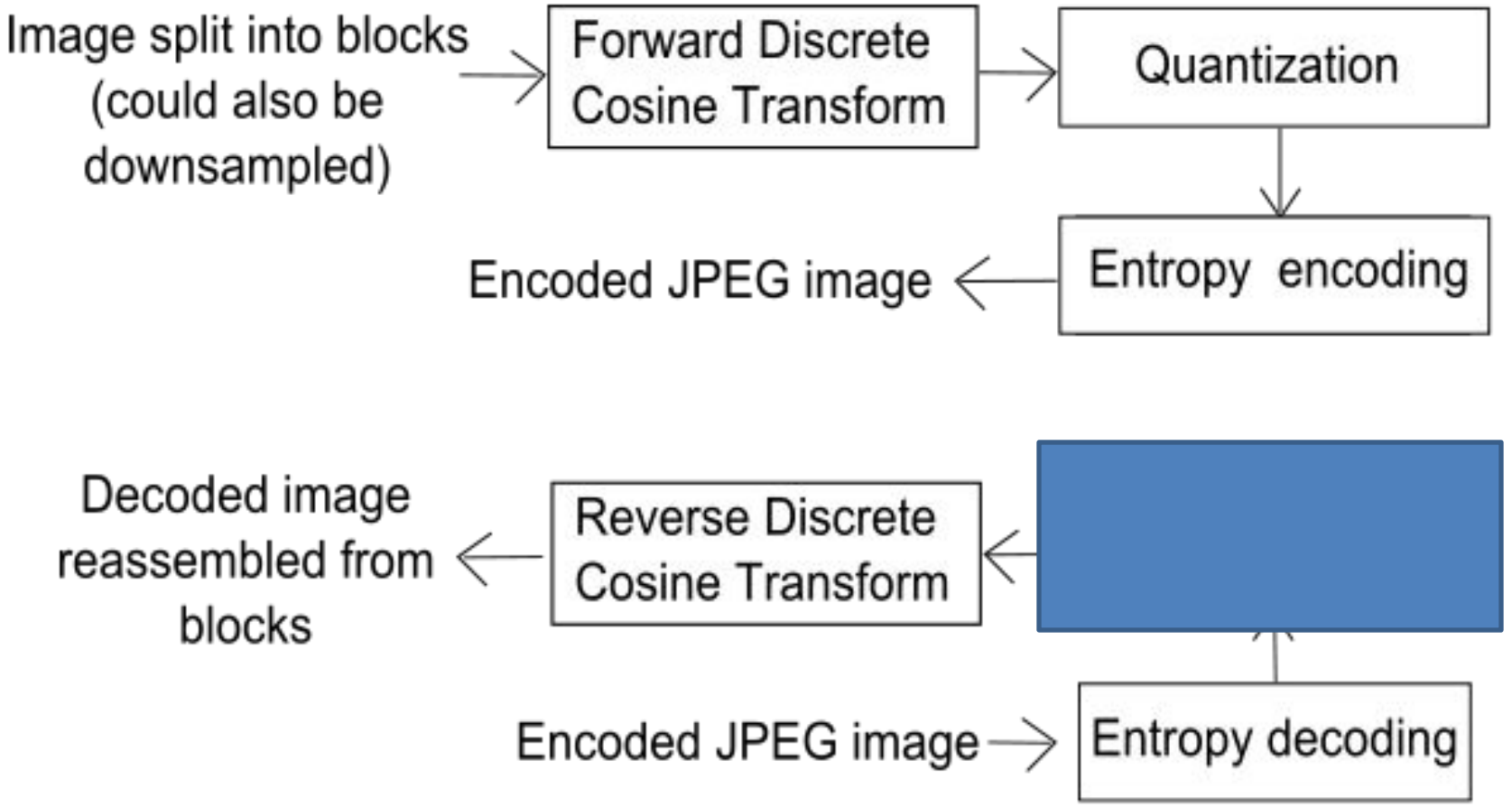


Image split into blocks (could also be downsampled)
Forward Discrete Cosine Transform
Quantization
Entropy encoding
Encoded JPEG image
Decoded image reassembled from blocks
Reverse Discrete Cosine Transform
Entropy decoding
Encoded JPEG image

STEP 1: Discrete Cosine Transform (DCT)

Discrete Cosine Transform (DCT) in 1D

$$F(u) = \sum_{n=0}^{N-1} f(n) a_N^{un}$$

$$f(n) = \sum_{u=0}^{N-1} F(u) \tilde{a}_N^{un}$$

$$DCT : a_N^{un} = \sqrt{\frac{1}{N}}, u = 0$$

$$a_N^{un} = \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(2n+1)u}{2N} \right), u = 1 \ldots N-1$$

$$\tilde{a}_N^{un} = a_N^{un}$$

$$DFT : a_N^{un} = e^{-j2\pi \frac{un}{N}}$$

$$\tilde{a}_N^{un} = a^{*un}{}_N \text{ (complex conjugate)}$$

Discrete Cosine Transform (DCT) in 1D

$$\begin{pmatrix} F(0) \\ \cdot \\ \cdot \\ \cdot \\ F(N-1) \end{pmatrix} = \begin{pmatrix} a_N^{0,0} & \cdots & a_N^{0,N-1} \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ a_N^{N-1,0} & \cdots & a_N^{N-1,N-1} \end{pmatrix} \begin{pmatrix} f(0) \\ \cdot \\ \cdot \\ \cdot \\ f(N-1) \end{pmatrix}$$

$$\mathbf{A} \in R^{N \times N}$$

$$\mathbf{A}\mathbf{A}^T = \mathbf{I}$$

$$\begin{pmatrix} f(0) \\ \cdot \\ \cdot \\ \cdot \\ f(N-1) \end{pmatrix} = \begin{pmatrix} a_N^{0,0} & \cdots & a_N^{N-1,0} \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ a_N^{0,N-1} & \cdots & a_N^{N-1,N-1} \end{pmatrix} \begin{pmatrix} F(0) \\ \cdot \\ \cdot \\ \cdot \\ F(N-1) \end{pmatrix}$$

$$\tilde{\mathbf{A}} \in R^{N \times N} \; (DCT \text{ Basis Matrix})$$

$$\tilde{\mathbf{A}}\tilde{\mathbf{A}}^T = \mathbf{I}$$

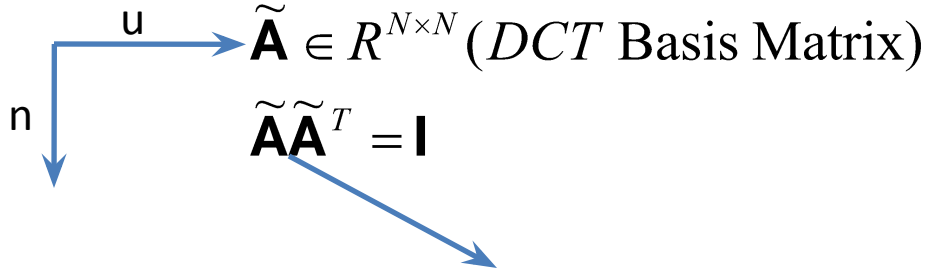
$$DCT : \tilde{\mathbf{A}} = \mathbf{A}^T$$

$$DFT : \tilde{\mathbf{A}} = \mathbf{A}^{*T} \text{ (conjugate transpose)}$$

DCT

- Expresses a signal as a linear combination of **cosine** bases (as opposed to the complex exponentials as in the Fourier transform).
- The coefficients of this linear combination are called **DCT coefficients**.
- Is **real-valued** unlike the Fourier transform!
- Discovered by Ahmed, Natarajan and Rao (1974)

$$\begin{pmatrix} f(0) \\ . \\ . \\ . \\ f(N-1) \end{pmatrix} = \begin{pmatrix} a_N^{0,0} & . & . & . & a_N^{0,N-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ a_N^{N-1,0} & . & . & . & a_N^{N-1,N-1} \end{pmatrix} \begin{pmatrix} F(0) \\ . \\ . \\ . \\ F(N-1) \end{pmatrix}$$



- DCT basis matrix is orthonormal. The dot product of any row (or column) with itself is 1. The dot product of any two different rows (or two different columns) is 0. The inverse is equal to the transpose.

- Being orthonormal, it preserves the squared norm, i.e. $\|\mathbf{f}\|^2 = \|\mathbf{F}\|^2$

- **DCT is NOT the real part of the Fourier!**

- DCT basis matrix is **NOT** symmetric.

- Columns of the DCT matrix are called the **DCT basis vectors**.

Digression: matrix view of a discrete orthonormal transform (Fourier transform used as example here)

- Remember:

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi xu/M}, 0 \le x \le M-1, 0 \le u \le M-1$$

- In matrix form, we write:

Fourier matrix: in any row, the value of *x* is fixed, the value of *u* ranges from 0 to *M*-1

$$\mathbf{f} = \mathbf{VF}, \mathbf{f} \in \mathbf{C}^M, \mathbf{F} \in \mathbf{C}^M, \mathbf{V} \in \mathbf{C}^{M \times M}; \mathbf{C} = \text{complex domain}$$

$$\mathbf{V} = \begin{pmatrix} e^{j2\pi(0)(0)/M} & e^{j2\pi(0)(1)/M} & \cdots & e^{j2\pi(0)(M-1)/M} \\ e^{j2\pi(1)(0)/M} & e^{j2\pi(1)(1)/M} & \cdots & e^{j2\pi(1)(M-1)/M} \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ e^{j2\pi(M-1)(0)/M} & e^{j2\pi(M-1)(1)/M} & \cdots & e^{j2\pi(M-1)(M-1)/M} \end{pmatrix},$$

$$V_{x'u'} = e^{j2\pi(x')(u')/M}$$

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi xu/M}, 0 \le x \le M-1, 0 \le u \le M-1$$

$$\mathbf{f} = \mathbf{VF}, \mathbf{f} \in \mathbf{C}^M, \mathbf{F} \in \mathbf{C}^M, \mathbf{V} \in \mathbf{C}^{M \times M}; \mathbf{C} = \text{complex domain}$$

$$\begin{pmatrix} f(0) \\ f(1) \\ . \\ . \\ f(M-1) \end{pmatrix} = \begin{pmatrix} e^{j2\pi(0)(0)/M} & e^{j2\pi(0)(1)/M} & . & . & e^{j2\pi(0)(M-1)/M} \\ e^{j2\pi(1)(0)/M} & e^{j2\pi(1)(1)/M} & . & . & e^{j2\pi(1)(M-1)/M} \\ . & . & . & . & . \\ . & . & . & . & . \\ e^{j2\pi(M-1)(0)/M} & e^{j2\pi(M-1)(1)/M} & . & . & e^{j2\pi(M-1)(M-1)/M} \end{pmatrix} \begin{pmatrix} F(0) \\ F(1) \\ . \\ . \\ F(M-1) \end{pmatrix}$$

DCT in 2D

$$F(u,v) = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} f(n,m)a_{NM}^{unvm}$$

$$f(n,m) = \sum_{u=0}^{N-1} F(u,v)\tilde{a}_{NM}^{unvm}$$

The DCT matrix is this case will have size *MN* x *MN*, and it will be the Kronecker product of two DCT matrices – one of size *M* x *M*, the other of size *N* x *N*. The DCT matrix for the 2D case is also orthonormal, it is NOT symmetric and it is NOT the real part of the 2D DFT.

$$DCT:$$

$$a_{NM}^{unvm} = \alpha(u)\alpha(v)\cos\left(\frac{\pi(2n+1)u}{2N}\right)\cos\left(\frac{\pi(2m+1)v}{2M}\right), u = 0\ldots N-1, v = 0\ldots M-1$$

$$\alpha(u) = \sqrt{1/N}\ (u=0), \text{else } \alpha(u) = \sqrt{2/N}$$

$$\alpha(v) = \sqrt{1/M}\ (v=0), \text{else } \alpha(v) = \sqrt{2/M}$$

$$\tilde{a}_{NM}^{unvm} = a_{NM}^{unvm}$$

What is a 2D Fourier Matrix?

- It is of the following form:

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)}$$

$$= \sum_{u=0}^{M-1} e^{j2\pi(ux/M)} \sum_{v=0}^{N-1} F(u,v)e^{j2\pi(vy/N)}$$

$$\mathbf{f} = \mathbf{VF}, \mathbf{f} \in \mathbf{C}^{MN}, \mathbf{F} \in \mathbf{C}^{MN}, \mathbf{V} \in \mathbf{C}^{MN \times MN}$$

$$\mathbf{V} = \mathbf{V}^{(1)} \otimes \mathbf{V}^{(2)}, \text{where } \mathbf{V}^{(1)} \text{and } \mathbf{V}^{(2)} \text{ are 1D Fourier matrices defined as :}$$

$$V_{ux}^{(1)} = e^{j2\pi ux/M}, V_{vy}^{(2)} = e^{j2\pi vy/N}, 0 \le u, x \le M-1, 0 \le v, y \le N-1$$

Equivalently, we can write

$$\mathbf{f}^{\mathbf{2D}} = \mathbf{V}^{(1)}\mathbf{F}^{\mathbf{2D}}\mathbf{V}^{(2)}$$

$$\mathbf{f}^{\mathbf{2D}} \in \mathbf{C}^{M \times N}, \mathbf{F}^{\mathbf{2D}} \in \mathbf{C}^{M \times N}, \mathbf{V}^{(1)} \in \mathbf{C}^{M \times M}, \mathbf{V}^{(2)} \in \mathbf{C}^{M \times M}$$

What is a 2D Fourier Matrix?

$$\mathbf{V} = \mathbf{V}^{(1)} \otimes \mathbf{V}^{(2)} \text{is the matrix Kronecker product defined as :}$$

$$\mathbf{V} = \begin{pmatrix} V^{(1)}(0,0)\mathbf{V}^{(2)} & . & . & . & V^{(1)}(0,M-1)\mathbf{V}^{(2)} \\ . & & . & . & . \\ . & & . & . & . \\ . & & . & . & . \\ V^{(1)}(M-1,0)\mathbf{V}^{(2)} & . & . & . & V^{(1)}(M-1,M-1)\mathbf{V}^{(2)} \end{pmatrix}$$

Consider a matrix **A** of size N_1 x N_2 and a matrix **B** of size M_1 x M_2. The size of their Kronecker product is given by N_1 M_1 x N_2 M_2. The Kronecker product is constructed by creating a rectangular grid of size N_1 x N_2 . In each cell of the grid, you place **B**. The copy of **B** in the cell at grid location *(i,j)* is multiplied by A_{ij}.

To understand why we compute a Kronecker product, we will work out a simple example for a 2 x 2 image, i.e. M = N = 2

$$f(0,0) = F(0,0)e^{i2\pi(0.0+0.0)/2} + F(0,1)e^{i2\pi(0.0+1.0)/2} + F(1,0)e^{i2\pi(1.0+0.0)/2} + F(1,1)e^{i2\pi(1.0+1.0)/2}$$

$$f(0,1) = F(0,0)e^{i2\pi(0.0+0.1)/2} + F(0,1)e^{i2\pi(0.0+1.1)/2} + F(1,0)e^{i2\pi(1.0+0.1)/2} + F(1,1)e^{i2\pi(1.0+1.1)/2}$$

$$f(1,0) = F(0,0)e^{i2\pi(0.1+0.0)/2} + F(0,1)e^{i2\pi(0.1+1.0)/2} + F(1,0)e^{i2\pi(1.1+0.0)/2} + F(1,1)e^{i2\pi(1.1+1.0)/2}$$

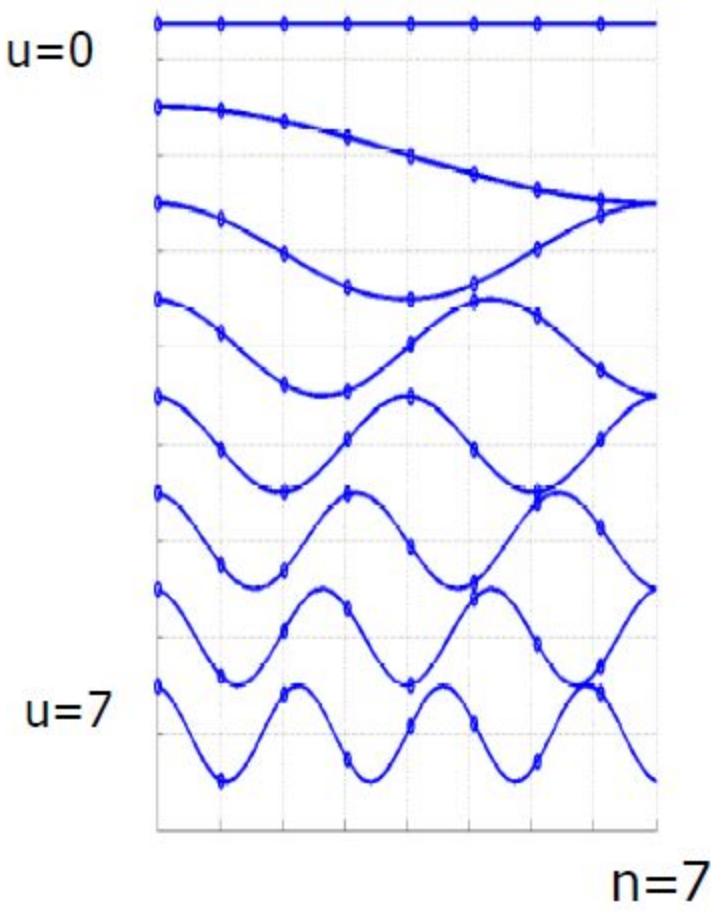
$$f(1,1) = F(0,0)e^{i2\pi(0.1+0.1)/2} + F(0,1)e^{i2\pi(0.1+1.1)/2} + F(1,0)e^{i2\pi(1.1+0.1)/2} + F(1,1)e^{i2\pi(1.1+1.1)/2}$$

$$\begin{pmatrix} f(0,0) \\ f(0,1) \\ f(1,0) \\ f(1,1) \end{pmatrix} = \begin{pmatrix} e^{i2\pi(0.0+0.0)/2} & e^{i2\pi(0.0+1.0)/2} & e^{i2\pi(1.0+0.0)/2} & e^{i2\pi(1.0+1.0)/2} \\ e^{i2\pi(0.0+0.1)/2} & e^{i2\pi(0.0+1.1)/2} & e^{i2\pi(1.0+0.1)/2} & e^{i2\pi(1.0+1.1)/2} \\ e^{i2\pi(0.1+0.0)/2} & e^{i2\pi(0.1+1.0)/2} & e^{i2\pi(1.1+0.0)/2} & e^{i2\pi(1.1+1.0)/2} \\ e^{i2\pi(0.1+0.1)/2} & e^{i2\pi(0.1+1.1)/2} & e^{i2\pi(1.1+0.1)/2} & e^{i2\pi(1.1+1.1)/2} \end{pmatrix} \begin{pmatrix} F(0,0) \\ F(0,1) \\ F(1,0) \\ F(1,1) \end{pmatrix}$$

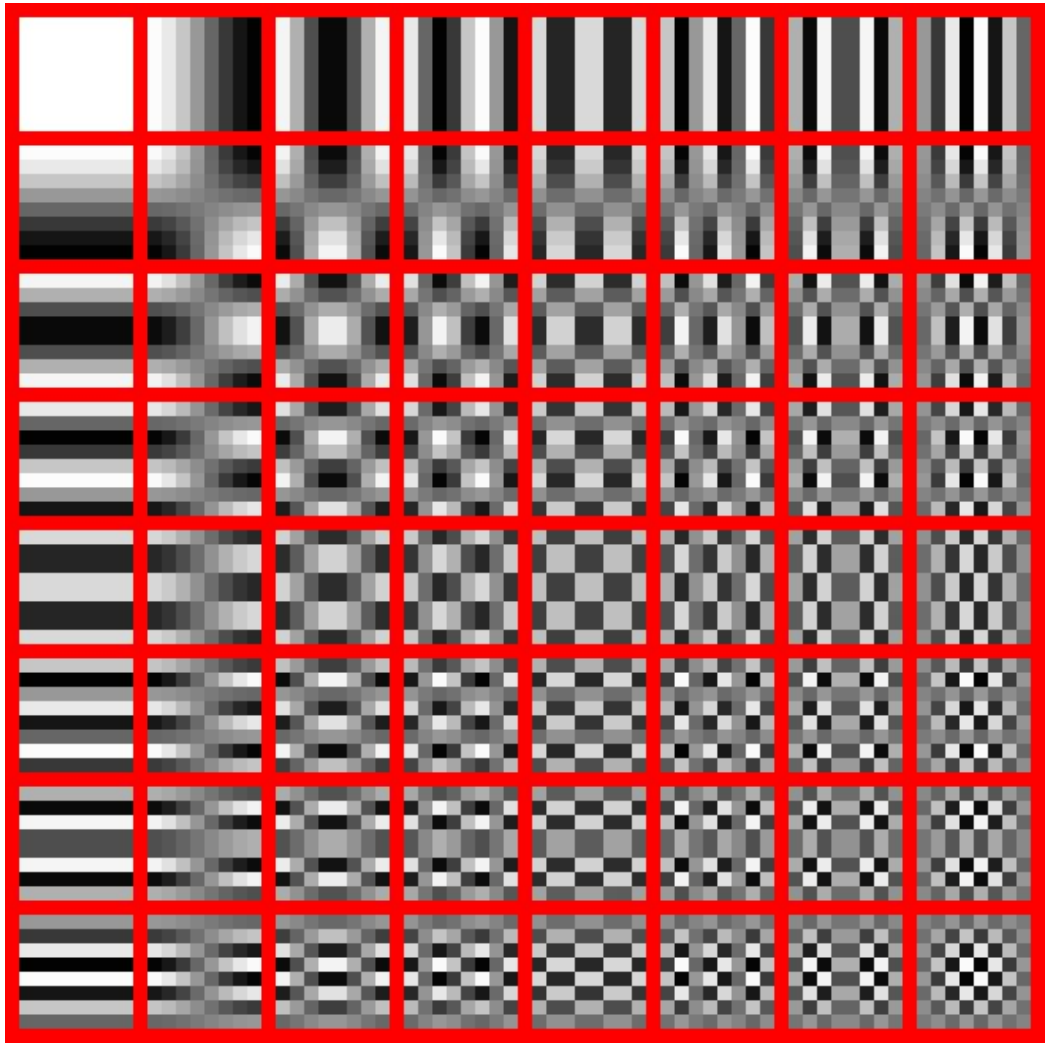
$$= \begin{pmatrix} e^{i2\pi(0.0)/2}\begin{bmatrix} e^{i2\pi(0.0)/2} & e^{i2\pi(1.0)/2} \\ e^{i2\pi(0.1)/2} & e^{i2\pi(1.1)/2} \end{bmatrix} & e^{i2\pi(1.0)}\begin{bmatrix} e^{i2\pi(0.0)/2} & e^{i2\pi(1.0)/2} \\ e^{i2\pi(0.1)/2} & e^{i2\pi(1.1)/2} \end{bmatrix} \\ e^{i2\pi(0.1)}\begin{bmatrix} e^{i2\pi(0.0)/2} & e^{i2\pi(1.0)/2} \\ e^{i2\pi(0.1)/2} & e^{i2\pi(1.1)/2} \end{bmatrix} & e^{i2\pi(1.1)}\begin{bmatrix} e^{i2\pi(0.0)/2} & e^{i2\pi(1.0)/2} \\ e^{i2\pi(0.1)/2} & e^{i2\pi(1.1)/2} \end{bmatrix} \end{pmatrix} \begin{pmatrix} F(0,0) \\ F(0,1) \\ F(1,0) \\ F(1,1) \end{pmatrix}$$

This big matrix is nothing but the Kronecker product of two 2 x 2 Fourier matrices.

How do the DCT bases look like? (1D case)



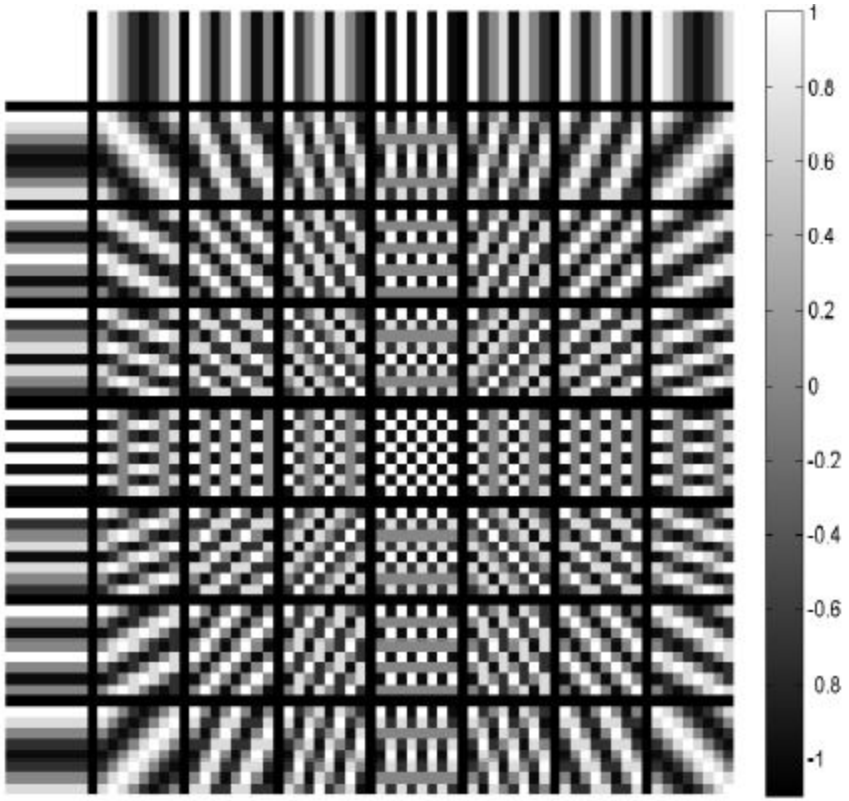
How do the DCT bases look like? (2D-case)



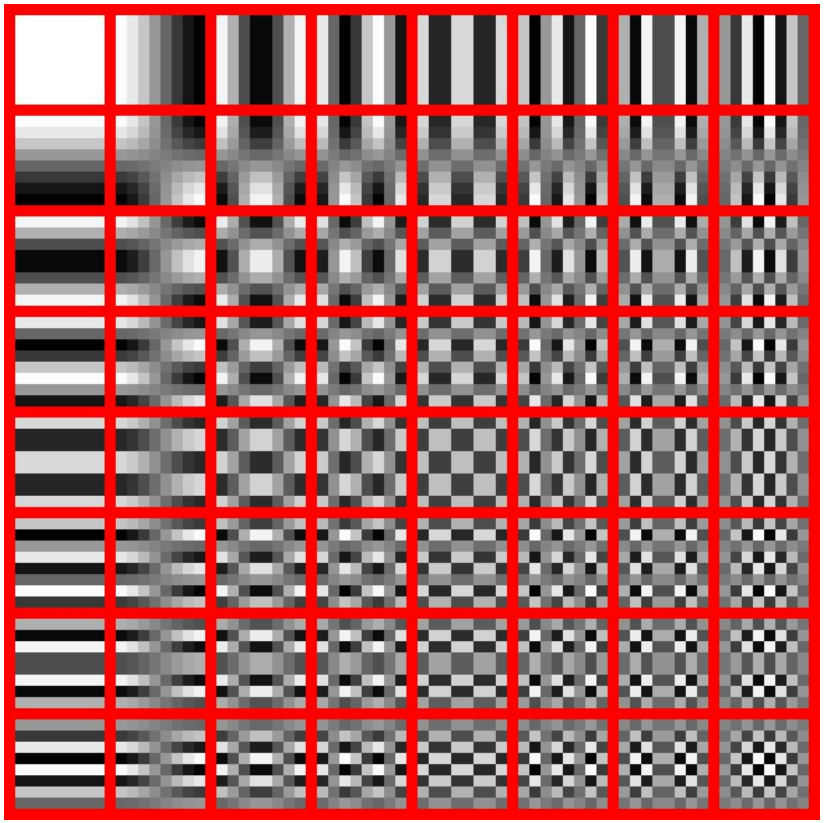
The DCT transforms an 8×8 block of input values to a linear combination of these 64 patterns. The patterns are referred to as the two-dimensional DCT *basis vectors*, and the output values are referred to as *transform coefficients*. Here each basis vector is reshaped to form an image.

http://en.wikipedia.org/wiki/JPEG

Again: DCT is NOT the real part of the DFT



Real part of DFT



DCT

http://en.wikipedia.org/wiki/JPEG

DCT on grayscale image patches

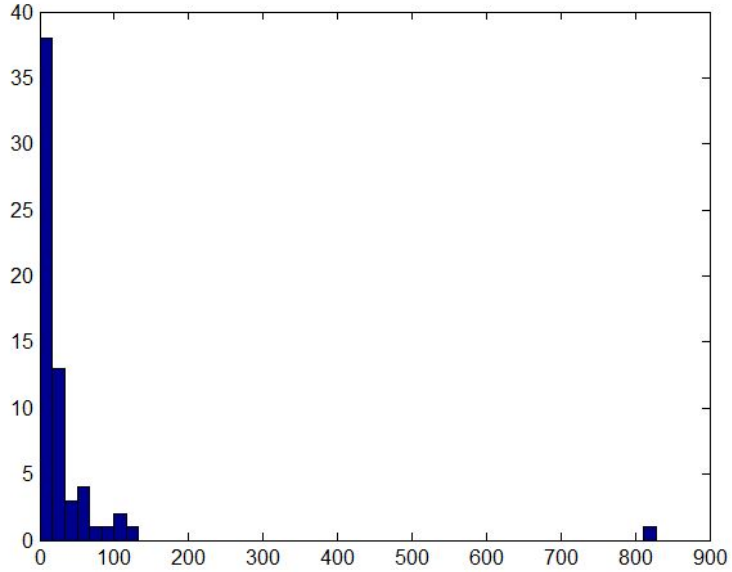
- The DCT coefficients of natural image patches have an amazing property.
- It is observed that most of the signal energy is concentrated in only a small number of coefficients.
- This is good news for compression! Store only a few coefficients, and throw away the rest.
- The corresponding error will be small, due to the orthonormal nature of the DCT.

149	74	92	74	74	74	149	162
87	74	117	30	74	105	180	130
30	117	105	43	105	130	149	105
74	162	105	74	105	117	105	105
117	149	74	117	74	105	74	149
149	87	74	87	74	74	117	180
105	74	105	43	61	117	180	149
74	74	105	74	105	130	149	105

IMAGE PATCH

828.3750	-106.7827	126.4183	-8.2540	-57.3750	-0.5311	-2.1682	29.8472
-6.0004	2.5328	8.3779	-7.1377	-17.3419	-6.9695	-11.1366	22.7612
-6.5212	-56.2336	23.5930	16.3746	-5.5436	74.2016	23.1543	65.2328
17.2141	29.9058	91.3782	-19.9119	106.2541	37.4804	15.8409	-25.1828
14.1250	53.2562	-30.5477	-0.8891	30.8750	-23.2787	-9.4005	-41.8019
5.7938	-2.9468	10.0191	2.8929	-16.5056	-2.4595	-5.1284	12.7364
-3.6579	2.3417	-14.8457	-0.7304	34.6327	-10.3257	-7.3430	-5.6082
-1.7071	-9.8264	-6.4722	-1.3611	-10.5811	-4.5081	-0.4332	-20.6615

DCT COEFFICIENTS



HISTOGRAM OF DCT COEFFICIENTS

Original image

Image reconstructed after discarding all DCT coefficients of non-overlapping 8 x 8 patches with absolute value less than 10, and then computing inverse DCT

Image reconstructed after discarding all DCT coefficients of non-overlapping 8 x 8 patches with absolute value less than 20, and then computing inverse DCT

Number of DCT coefficients of non-overlapping 8 x 8 patches with absolute value less than 10 was 34,377 out of a total of 65536 (64 coefficients for each 8 x 8 patch, totally 1024 such patches). **This is more than 50%.** Corresponding percentage for DFT was 1%.

Number of DCT coefficients of non-overlapping 8 x 8 patches with absolute value less than 20 was 51,045 out of a total of 65536 (64 coefficients for each 8 x 8 patch, totally 1024 such patches). This is more than 78%. Corresponding percentage for DFT was 7%.

Why DCT? DFT and DCT comparison

	DFT	DCT
Orthonormal	**Yes**	**Yes**
Real/complex	**Complex**	**Real**
Separable in 2D	**Yes**	**Yes**
Norm-preserving	**Yes**	**Yes**
Inverse exists	**Yes**	**Yes**
Fast implementation	**Yes (fft)**	**Yes (uses fft)**
Energy compaction for natural images	**Good/Fair**	**Much Better**

DCT has better energy compaction than DFT because…

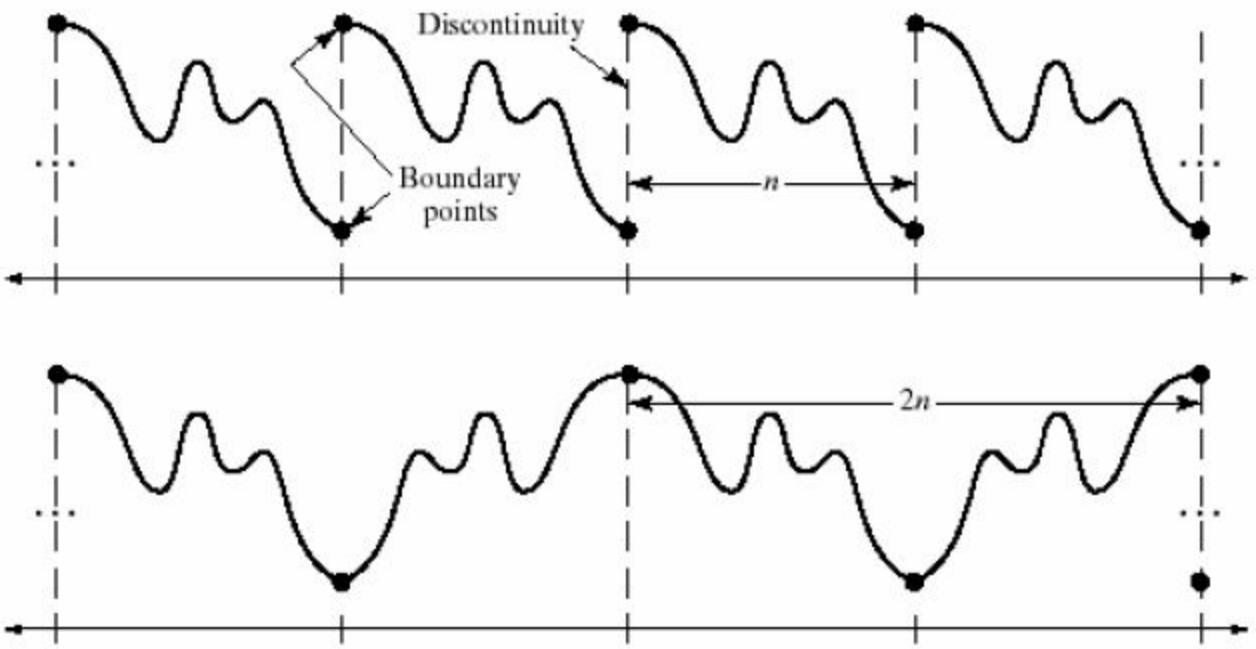
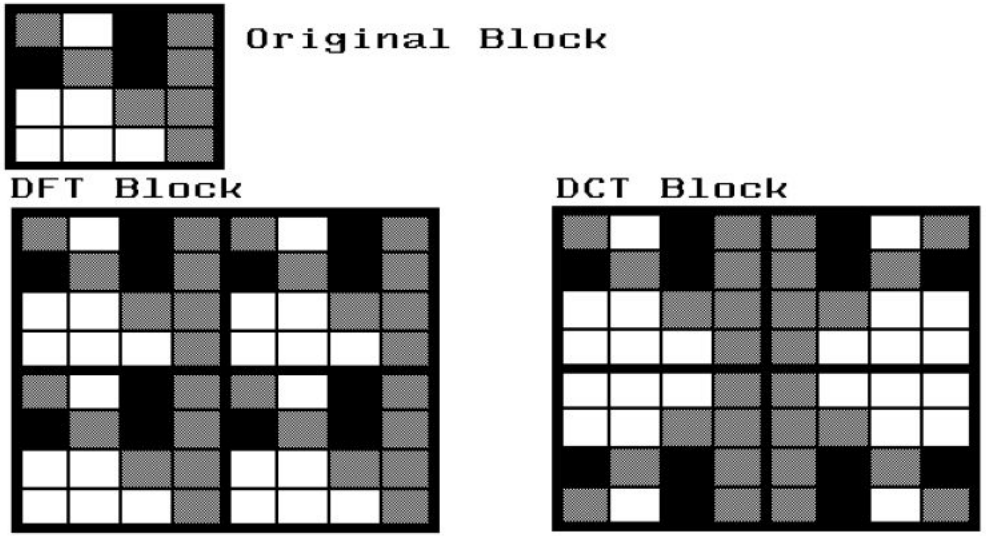


FIGURE 8.32 The periodicity implicit in the 1-D (a) DFT and (b) DCT.

Recall that the DFT of a sequence is equal to the Discrete Fourier Series (DFS) of a periodic extension of that sequence. In computing the DFT of a signal of length *n*, there is the implicit extension of several copies of the signal placed one after the other (*n*-point periodicity). The resultant discontinuities require several frequencies for good representation in the DFS. As against this, the discontinuities are reduced in a DCT because a reflected copy of the signal is appended to it (2*n*-point periodicity) before computing the DFS.

DCT has better energy compaction than DFT because…



Arrangement of pixel blocks in discrete Fourier transform (DFT) and discrete cosine transform (DCT). Pixel blocks in the DFT are strictly periodic, thus possibly producing large discontinuities in gray value at the edges; pixel blocks in the DCT are reversed, replicating gray values across the edges of blocks and reducing the discontinuities at the edges.

DCT computational complexity

- Naïve implementation (matrix times vector) is $O(N^2)$ for a vector of N elements.
- You can speed this up to $O(N \log N)$ using the FFT as shown on next slide.

$$\tilde{f}(n) = f(n), 0 \le n \le N-1$$

$$\tilde{f}(n) = f(2N-n-1), N \le n \le 2N-1$$

Reflected version of f, appended to f.

It can be shown that :

$$DCT(f)(u) = F(u) = e^{-j\pi u/(2N)} DFT(\tilde{f})(u), 0 \le u \le N-1$$

(with some caveats – next slide)

DCT of **f** is computed from the DFT of the sequence $\tilde{\mathbf{f}}$ of double length as **f**. Only the first N frequencies are picked.

In MATLAB, you have the commands called dct and idct (in 1D) and dct2 and idct2 (in 2D).

supporting code:
https://www.cse.iitb.ac.in/~ajitvr/CS663_Fall2024/Code_Compression/dct_dft_relation.m

$$DFT(\tilde{f})(u) = \sum_{n=0}^{N-1} f(n)e^{-j2\pi un/2N} + \sum_{n=N}^{2N-1} f(2N-n-1)e^{-j2\pi un/2N}$$

$$= \sum_{n=0}^{N-1} f(n)e^{-j2\pi un/2N} + \sum_{n=0}^{N-1} f(n)e^{-j2\pi u(2N-n-1)/2N}$$

$$= \sum_{n=0}^{N-1} f(n)\left(e^{-j2\pi un/2N} + e^{-j2\pi u(2N-n-1)/2N}\right)$$

$$= \sum_{n=0}^{N-1} f(n)\left(e^{-j2\pi un/2N}e^{-j2\pi u/4N}e^{j2\pi u/4N} + e^{-j2\pi u(2N)/2N}e^{j2\pi un/2N}e^{j2\pi u/2N}\right)$$

1

$$= \sum_{n=0}^{N-1} f(n)\left(e^{-j2\pi un/2N}e^{-j2\pi u/4N}e^{j2\pi u/4N} + e^{j2\pi un/2N}e^{j2\pi u/2N}\right)$$

$$= e^{j2\pi u/4N}\sum_{n=0}^{N-1} f(n)\left(e^{-j2\pi un/2N}e^{-j2\pi u/4N} + e^{j2\pi un/2N}e^{j2\pi u/4N}\right)$$

$$= e^{j2\pi u/4N}\sum_{n=0}^{N-1} 2f(n)\left(\frac{e^{-j2\pi u(n+1/2)/2N} + e^{j2\pi u(n+1/2)/2N}}{2}\right)$$

$$= e^{j2\pi u/4N}\sum_{n=0}^{N-1} 2f(n)\cos\left(\frac{\pi u(2n+1)}{2N}\right)$$

$$\therefore \sum_{n=0}^{N-1} 2f(n)\cos\left(\frac{\pi u(2n+1)}{2N}\right) = e^{-j2\pi u/4N}DFT(\tilde{f})(u)$$

See code in google drive folder

http://en.wikipedia.org/wiki/Discrete_cosine_transform
http://www.ecsutton.ece.ufl.edu/dip/handouts/dct.pdf

You would noticed that the constant factors sqrt(1/N) and sqrt(2/N) are missing in this expression. These factors are essential for the DCT matrix to be orthonormal, but their presence doesn't allow for this relationship between DCT and DFT.

Which is the best orthonormal basis?

- Consider a set of *M* data-points (e.g. image patches in a vectorized form) represented as a linear combination of column vectors of an ortho-normal basis matrix:

$$\mathbf{q}_i = \mathbf{U}\boldsymbol{\theta}_i, \mathbf{q}_i \in R^{N\times 1}, \boldsymbol{\theta}_i \in R^{N\times 1}, \mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

- Suppose we reconstruct each patch using only a subset of some *k* coefficients as follows:

$$\tilde{\mathbf{q}}_i^{(k)} = \mathbf{U}\tilde{\boldsymbol{\theta}}_i, \text{ where } \tilde{\boldsymbol{\theta}}_i \text{ is obtained by setting to 0 all}$$

$$\text{except } k \text{ coefficients (the same } k \text{ coefficients are retained for all patches)}$$

Which is the best orthonormal basis?

- For which orthonormal basis **U** is the following error the lowest:

$$E(\mathbf{U}) = \sum_{i=1}^{M} \left\| \tilde{\mathbf{q}}_i^{(k)} - \mathbf{q}_i \right\|^2$$

Which is the best orthonormal basis?

- The answer is the PCA basis, i.e. the set of *k* eigenvectors of the correlation matrix **C**, corresponding to the *k* largest eigen-values. Here is **C** is defined as:

$$\mathbf{C} = \frac{1}{M-1}\sum_{i=1}^{M}\mathbf{q}_i\mathbf{q}_i^T,$$

$$C_{kl} = \frac{1}{M-1}\sum_{i=1}^{M}q_{ikl}q_{ilk}$$

PCA: separable 2D version

- Find the correlation matrix $\mathbf{C}_R$ of row vectors from the patches.
- Find the correlation matrix $\mathbf{C}_C$ of column vectors from the patches.
- The final PCA basis is the Kronecker product of the individual bases:

$$\mathbf{C}_R = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{n} q_i(j,:)'\, q_i(j,:);$$

$$[\mathbf{V}_R, \mathbf{D}_R] = eig(\mathbf{C}_R); q_i(j,:) \in R^{1\times n} - j^{th} \text{ row vector of } \mathbf{q_i}$$

$$\mathbf{C}_C = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{n} q_i(:,j) q_i(:,j)';$$

$$[\mathbf{V}_C, \mathbf{D}_C] = eig(\mathbf{C}_C); q_i(:,j) \in R^{n\times 1} - j^{th} \text{ column vector of } \mathbf{q_i}$$

$$\mathbf{V} = \mathbf{V}_R \otimes \mathbf{V}_C; \mathbf{V}\mathbf{V}^T = I; \mathbf{V} \in R^{n^2\times n^2}, \mathbf{V}_R \in R^{n\times n}, \mathbf{V}_C \in R^{n\times n}, \mathbf{q_i} \in R^{n\times n}$$

But PCA is not used in JPEG, because…

- It is image-dependent, and the basis matrix would need to be computed afresh for each image.
- The basis matrix would need to be **stored** for each image.
- It is **expensive** to compute – $O(n^3)$ for a vector with n elements.

The DCT is used instead!

DCT and PCA

- DCT can be computed very fast using fft.
- It is universal – no need to store the DCT bases explicitly.
- DCT has very good energy compaction properties, only slightly worse than PCA.

Code:
https://www.cse.iitb.ac.in/~ajitvr/CS663_Fall2024/Code_Compression/dct_pca.m

Experiment

- Suppose you extract M ~ 100,000 small-sized (8 x 8) patches from a set of images.
- Compute the column-column and row-row correlation matrices.

$$\mathbf{C_C} = \frac{1}{M-1}\sum_{i=1}^{M}\mathbf{P}_i\mathbf{P}_i^T = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{8}P_i(:,j)P_i(:,j)';$$

$$\mathbf{C_R} = \frac{1}{M-1}\sum_{i=1}^{M}\mathbf{P}_i^T\mathbf{P}_i = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{8}P_i(j,:)'P_i(j,:);$$

- Compute their eigenvectors $\mathbf{V}_R$ and $\mathbf{V}_C$.
- The eigenvectors will be very similar to the columns of the 1D-DCT matrix! (as evidenced by dot product values).
- Now compute the Kronecker product of $\mathbf{V}_R$ and $\mathbf{V}_C$ and call it $\mathbf{V}$. Reshape each column of $\mathbf{V}$ to form an image. These images will appear very similar to the DCT bases.

0.3536	0.4904	0.4619	0.4157	0.3536	0.2778	0.1913	0.0975
0.3536	0.4157	0.1913	-0.0975	-0.3536	-0.4904	-0.4619	-0.2778
0.3536	0.2778	-0.1913	-0.4904	-0.3536	0.0975	0.4619	0.4157
0.3536	0.0975	-0.4619	-0.2778	0.3536	0.4157	-0.1913	-0.4904
0.3536	-0.0975	-0.4619	0.2778	0.3536	-0.4157	-0.1913	0.4904
0.3536	-0.2778	-0.1913	0.4904	-0.3536	-0.0975	0.4619	-0.4157
0.3536	-0.4157	0.1913	0.0975	-0.3536	0.4904	-0.4619	0.2778
0.3536	-0.4904	0.4619	-0.4157	0.3536	-0.2778	0.1913	-0.0975

DCT matrix: dctmtx command from MATLAB (see code on website)

0.3517	-0.4493	-0.4278	0.4230	0.3754	0.3247	-0.2250	-0.1245
0.3534	-0.4366	-0.2276	-0.0110	-0.3078	-0.4746	0.4732	0.2975
0.3543	-0.3101	0.1728	-0.4830	-0.3989	0.0498	-0.4299	-0.4109
0.3546	-0.1115	0.4799	-0.3005	0.3342	0.4102	0.1856	0.4761
0.3547	0.1141	0.4823	0.2944	0.3301	-0.4182	0.1745	-0.4771
0.3543	0.3104	0.1771	0.4834	-0.3977	-0.0322	-0.4308	0.4103
0.3535	0.4357	-0.2319	0.0143	-0.3009	0.4656	0.4851	-0.2975
0.3520	0.4468	-0.4328	-0.4204	0.3686	-0.3253	-0.2342	0.1261

$\mathbf{V}_C$: Eigenvectors of column-column correlation matrix

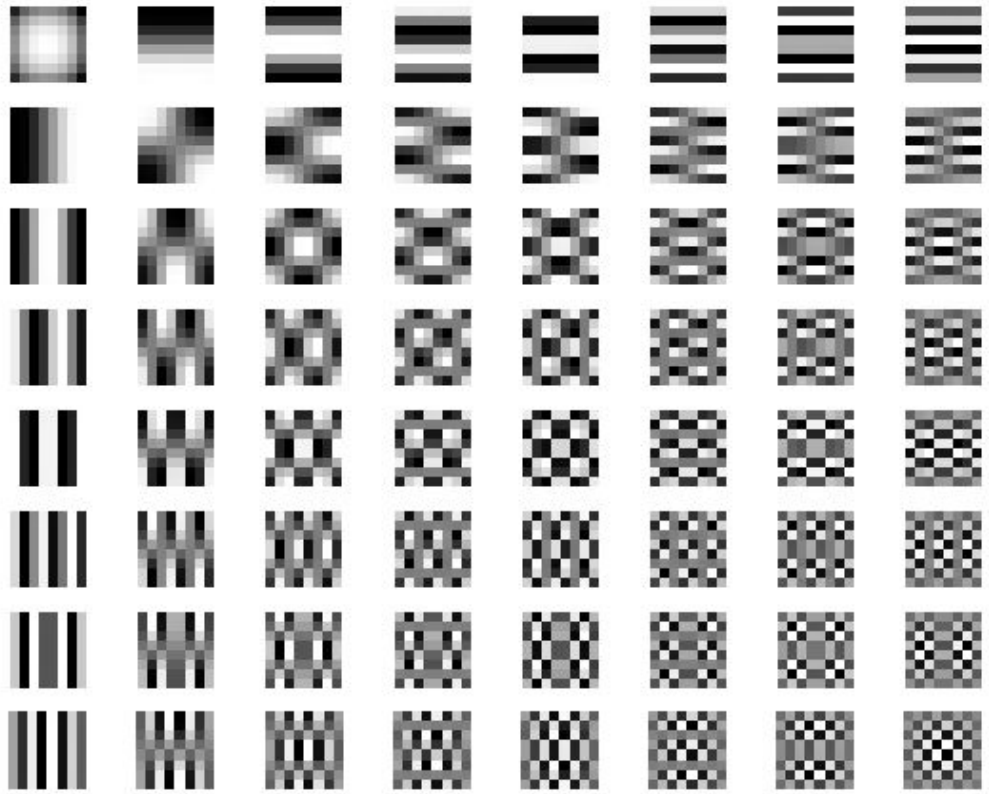
0.3520	-0.4461	-0.4305	0.4224	0.3696	0.3247	0.2342	0.1283
0.3537	-0.4338	-0.2345	-0.0114	-0.3000	-0.4671	-0.4814	-0.3028
0.3545	-0.3086	0.1662	-0.4896	-0.4007	0.0359	0.4261	0.4102
0.3548	-0.1145	0.4763	-0.3031	0.3339	0.4198	-0.1800	-0.4713
0.3548	0.1056	0.4839	0.2926	0.3349	-0.4194	-0.1766	0.4733
0.3543	0.3043	0.1863	0.4833	-0.4028	-0.0354	0.4269	-0.4097
0.3532	0.4389	-0.2269	0.0180	-0.3008	0.4654	-0.4811	0.3037
0.3512	0.4562	-0.4300	-0.4126	0.3694	-0.3242	0.2335	-0.1319

$\mathbf{V}_R$: Eigenvectors of row-row correlation matrix

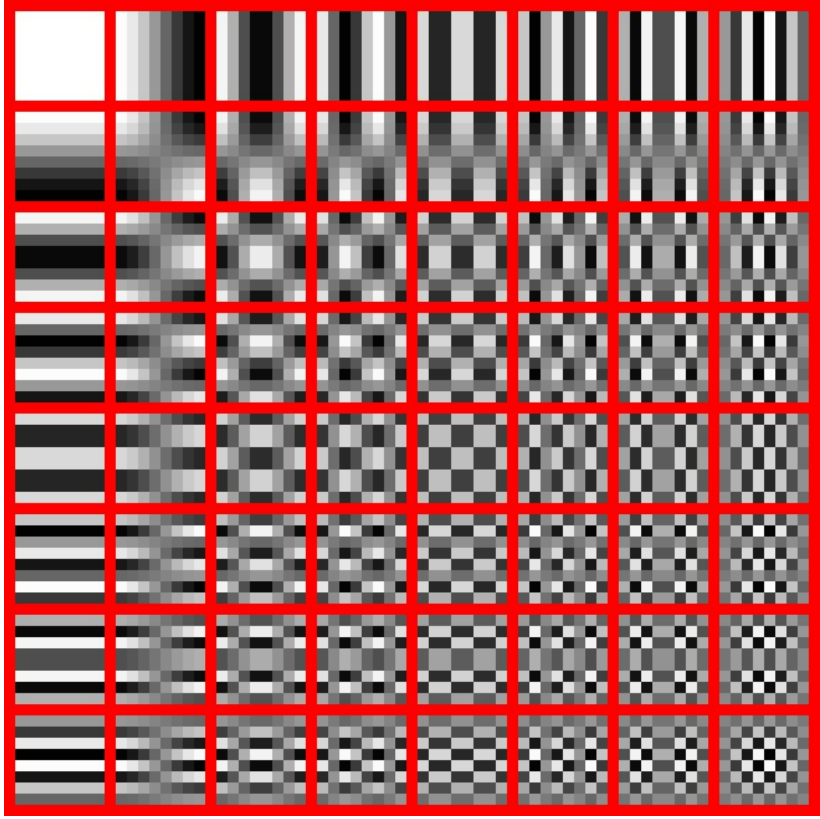
Absolute value of dot products between the columns of DCT matrix and columns of V_R (left) and V_C (right)

1.0000	0.0007	0.0032	0.0002	0.0013	0.0001	0.0005	0.0000
0.0007	**0.9970**	0.0097	0.0689	0.0009	0.0322	0.0003	0.0110
0.0033	0.0106	**0.9968**	0.0118	0.0713	0.0004	0.0334	0.0025
0.0002	0.0718	0.0124	**0.9926**	0.0007	0.0927	0.0017	0.0276
0.0010	0.0001	0.0737	0.0004	**0.9942**	0.0008	0.0780	0.0010
0.0000	0.0261	0.0015	0.0962	0.0005	**0.9934**	0.0011	0.0569
0.0003	0.0007	0.0276	0.0021	0.0802	0.0010	**0.9964**	0.0013
0.0000	0.0076	0.0026	0.0227	0.0012	0.0596	0.0015	**0.9979**

1.0000	0.0002	0.0029	0.0001	0.0010	0.0000	0.0004	0.0000
0.0002	**0.9965**	0.0028	0.0766	0.0005	0.0314	0.0009	0.0107
0.0029	0.0025	**0.9969**	0.0046	0.0728	0.0017	0.0304	0.0013
0.0001	0.0795	0.0044	**0.9923**	0.0029	0.0916	0.0015	0.0243
0.0008	0.0003	0.0747	0.0026	**0.9948**	0.0061	0.0696	0.0004
0.0000	0.0246	0.0021	0.0949	0.0069	**0.9940**	0.0131	0.0452
0.0003	0.0004	0.0252	0.0003	0.0715	0.0137	**0.9970**	0.0002
0.0000	0.0076	0.0013	0.0207	0.0001	0.0476	0.0009	**0.9986**



64 columns of **V** – each reshaped to form an 8 x 8 image, and rescaled to fit in the 0-1 range. Notice the similarity between the DCT bases and the columns of **V**. Again, **V** is the Kronecker product of $\mathbf{V}_R$ and $\mathbf{V}_C$.



DCT bases

DCT and PCA

- DCT is very close to PCA when the patches come from what is called as a **stationary first order Markov process**, i.e.

$$q_i = \rho q_{i-1} + \eta_i, \eta_i \sim N(0, \sigma_\eta^2), \rho < 1$$

$$E(q_i q_{i-1}) = \rho\sigma_q^2, E(q_i q_{i-2}) = \rho^2\sigma_q^2, \ldots, E(q_i q_{i-n+1}) = \rho^{n-1}\sigma_q^2,$$

$$\mathbf{C} = \sigma_q^2 \begin{pmatrix} 1 & \rho & \rho^2 & . & \rho^{n-1} \\ \rho & 1 & . & . & . \\ \rho^2 & . & . & . & . \\ . & . & . & . & . \\ \rho^{n-1} & . & . & . & 1 \end{pmatrix}, C_{ij} = E(q_i q_j), \sigma_q^2 = E(q_i^2) \text{ for any } i$$

DCT and PCA

- One can show that the eigenvectors of the correlation matrix of the form seen on the previous slide are the DCT basis vectors!

- Natural images approximate this first order Markov model, and hence DCT is almost as good as PCA for compression of a large ensemble of image patches.

- DCT has the advantage of being a universal basis and also the DCT coefficients are more efficiently computable than PCA coefficients (because DCT computation uses FFT).

More results from the previous experiment. See code:

https://www.cse.iitb.ac.in/~ajitvr/CS663_Fall2024/Code_Compression/dct_pca.m

1.0000	0.9902	0.9795	0.9733	0.9682	0.9639	0.9604	0.9570
0.9902	1.0005	0.9908	0.9795	0.9734	0.9684	0.9643	0.9605
0.9795	0.9908	1.0010	0.9908	0.9796	0.9735	0.9689	0.9646
0.9733	0.9795	0.9908	1.0005	0.9904	0.9793	0.9735	0.9686
0.9682	0.9734	0.9796	0.9904	1.0004	0.9903	0.9794	0.9734
0.9639	0.9684	0.9735	0.9793	0.9903	1.0001	0.9903	0.9793
0.9604	0.9643	0.9689	0.9735	0.9794	0.9903	1.0004	0.9904
0.9570	0.9605	0.9646	0.9686	0.9734	0.9793	0.9904	1.0002

CR/CR(1,1) -
Notice it can be approximated by the form shown two slides before, with $\rho \sim 0.99$

1.0000	0.9888	0.9770	0.9704	0.9648	0.9599	0.9554	0.9510
0.9888	1.0004	0.9891	0.9768	0.9703	0.9646	0.9596	0.9548
0.9770	0.9891	1.0004	0.9886	0.9764	0.9698	0.9640	0.9587
0.9704	0.9768	0.9886	0.9994	0.9878	0.9755	0.9687	0.9627
0.9648	0.9703	0.9764	0.9878	0.9986	0.9870	0.9746	0.9676
0.9599	0.9646	0.9698	0.9755	0.9870	0.9978	0.9861	0.9734
0.9554	0.9596	0.9640	0.9687	0.9746	0.9861	0.9967	0.9847
0.9510	0.9548	0.9587	0.9627	0.9676	0.9734	0.9847	0.9951

CC/CC(1,1) -
Notice it can be approximated by the form shown two slides before, with $\rho \sim 0.9888$

Computation of DCT coefficients in JPEG

- Before computation, the value 128 (midpoint of the range 0 to 255) is subtracted from every pixel value.
- This changes the range of intensity values from 0 to 255, to -128 to 127.
- This also changes the range of DCT coefficient values from 0 to 2048, to -1024 to +1024.

STEP 2: Quantization

Quantization

- The DCT coefficients are floating point numbers and storing them in a file will produce no compression. So they need to be **quantized**.
- The human eye is not sensitive to changes in the higher frequency content.
- So we can have cruder quantization for the higher frequency coefficients and a finer one for the lower frequency coefficients.

Quantization

- Quantization is performed by dividing the DCT coefficient matrix *element-wise* by a quantization matrix and rounding off to the nearest integer.
- The quantization matrix on the next slide is for quality factor $Q = 50$.
- Matrices for *lower Q* values are obtained by scaling the $Q = 50$ matrix with a constant $50/Q$ – which *increases* the values in the quantization matrix.
- Matrices for *higher Q* values are obtained by scaling the $Q = 50$ matrix with a constant $50/Q$ – which *decreases* the values in the quantization matrix.

$$G = \begin{matrix} \\ \\ \end{matrix} \overset{u}{\longrightarrow} \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \downarrow v.$$

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

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Quantization matrix for Q = 50: notice the higher values in the matrix for higher frequency coefficients

$$B_{uv} = \mathrm{round}(G_{uv} / M_{uv})$$

Most of the values in B are 0!
They need not be stored! Only the non-zero values in B will be stored!

How was this quantization matrix picked?

- The quantization error is given by $e_{uv} = G_{uv} - B_{uv}M_{uv}$ where $B_{uv} = \text{round}(G_{uv} / M_{uv})$.
- The maximum possible value of the error is given by $M_{uv}/2$.
- Psychophysical studies have been performed to find threshold values of DCT coefficients, i.e. for each frequency (u,v), these studies have determined the smallest DCT coefficient value that yielded a visible signal. This threshold is called t_{uv}.
- We set $M_{uv} = 2t_{uv}$ so that the errors remain invisible.

STEP 3: Lossless compression steps: Huffman encoding and Run length encoding

Huffman encoding

- Input: a set of non-zero quantized DCT coefficients from all the different blocks of the image (values lying between -1024 to +1024).
- Output: a set of encoded coefficients with length (in terms of number of bits) less than that of the original set.
- Principles behind Huffman encoding:

(1) Encode the **more frequently** occurring coefficients with **fewer** bits. Encode the **rarely** occurring coefficients with **more** bits. This will reduce the average bit-length.

(2) Ensure that the encoding for no coefficient is a strict prefix of the encoding of any other coefficient (to be explained on next slide). This is called a **"prefix-free code".**

Huffman encoding example

- Consider a set of alphabets {*a,e,q*}**.** Let the frequency of an alphabet *x* be denoted as *p(x)*.
- Assume *p(e) > p(a) > p(q)* [actually true in the English language].
- Consider the following code-word assignment: *e – 0, a – 1, q – 01* (note: we assigned more bits for *q*). Now consider the encoded stream: 001. It can be interpreted as *'eea'* or *'eq'.*
- The reason for this ambiguity is that the code for *'e'* is a strict prefix of the code for *'q'.*
- For unambiguous decoding, we need prefix-free codes. Example *e – 0, a – 10, q – 11* is one example of a prefix-free code.

Huffman encoding example

- The Huffman encoding algorithm asks the following question:

  Given a set of *n* alphabets $A = \{a_i\}$ with corresponding frequencies $\{p(a_i)\}$ (each frequency lies from 0 to 1), what prefix-free encoding yields the least average bit length? That is, which set of code-words $\{\lambda(a_i)\}$ will minimize

$$L(\{\lambda(a_i)\}) = \sum_{i=1}^{n} p(a_i) \boxed{|\lambda(a_i)|}$$

Length of the code-word $\lambda(a_i)$

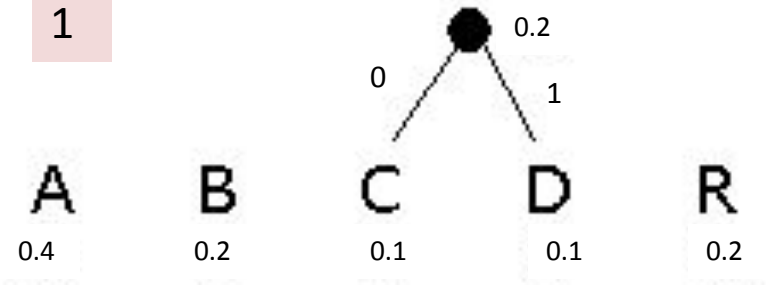
Algorithm

1. Sort alphabets in **increasing** order of frequency. Create a leaf node from each alphabet. These leaf nodes will belong to a binary tree called the **Huffman tree**.
2. Combine the two **lowest** frequency nodes s_1 and s_2 to create a parent node s_{12}. s_1 and s_2 will be the left and right child of s_{12}. The frequency of s_{12} is given by $\mathbf{p(s_{12}) = p(s_1) + p(s_2)}$.
3. Label the edge from s_{12} to s_1 with a '0' and the edge from s_{12} to s_2 with a '1'.
4. Delete s_1 and s_2 from the sorted list of alphabets and insert the node s_{12}, i.e. root node of the tree (s_{12},s_1,s_2) in the correct place depending on the value of $\mathbf{p(s_{12})}$.
5. Repeat steps 2 to 4 until there is only one node in the list. This will be the **root node of the final Huffman tree**.
6. Traverse the tree from the root node until each leaf and collect all the binary symbols along every edge into a string. This string will form the code word for that symbol.

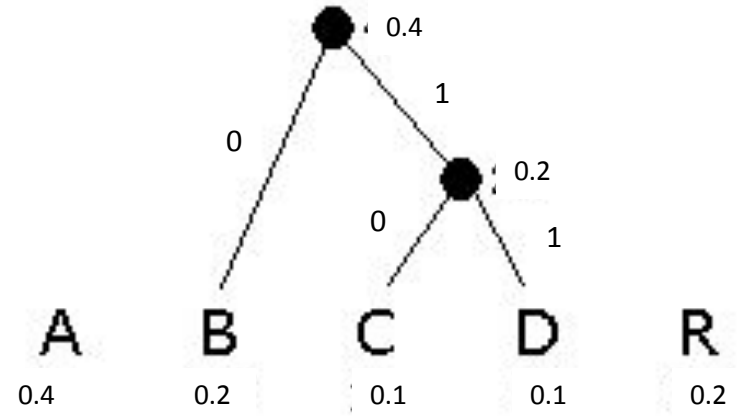
www.cis.upenn.edu/~matuszek/cit594-2002/Slides/**huffman**.ppt

Example

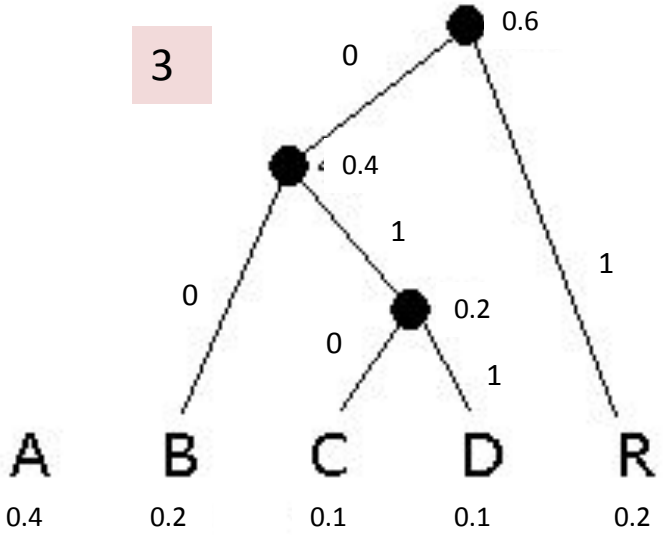
1



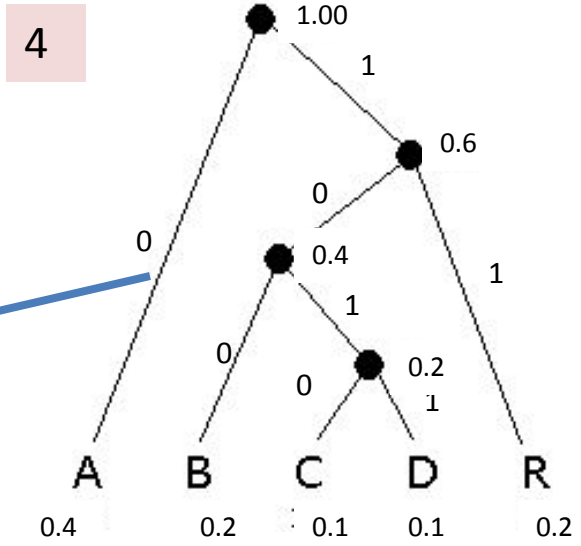
2



3

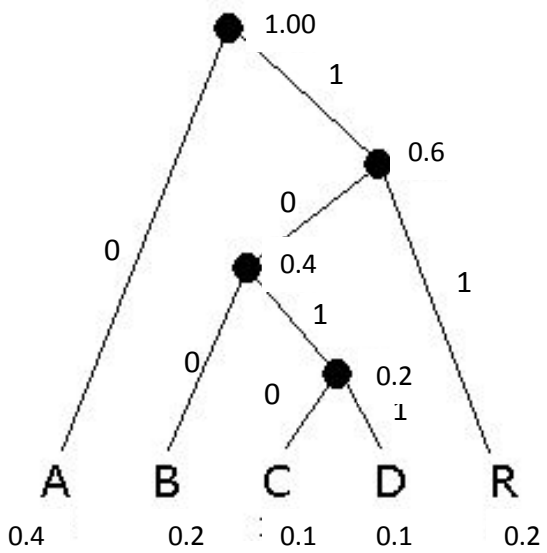


4



A = 0
B = 100
C = 1010
D = 1011
R = 11

This is a prefix-free code. No leaf node is on the path to any other node.



A = 0
B = 100
C = 1010
D = 1011
R = 11

Average code length here
= 1 * p(A) + 3 * p(B) + 4 * p(C) + 4*p(D) + 2 *p(R) = 0.4 + 0.6 + 0.4 + 0.4 + 0.4 = 2.2.

Input string: RABBCDR
Encoded bit stream:
11-0-**100-**100-**1010-**1011-11
Decoded string: RABBCDR

To perform encoding, we maintain an initially empty encoded bit stream. Read a symbol from the input, traverse the Huffman tree from root node to the leaf node for that symbol, collecting all the bit labels on the traversed path, and appending them to the encoded bit stream. Repeat this for every symbol from the input. Example: for R, we write 11.

To perform decoding, read the encoded bit stream, and traverse the Huffman tree from the root node toward a leaf node, following the path as indicated by the bit stream. For example, if you read in 11, you would travel to the leaf node R. When you reach a leaf node, append its associated symbol to the decoded output. Go back to the root node and traverse the tree as per the remaining bits from the encoded bit stream.

About the algorithm

- This is a greedy algorithm, which is guaranteed to produce the prefix-free code with **minimal average length** (proof beyond the scope of the course).
- There could be multiple sets of code words with the same average bit length. Huffman encoding produces one of them, depending on the order in which the nodes were combined, and the convention for labeling the edges with a 0 or a 1.

Huffman Trees and Entropy

- The average code length as computed by this algorithm satisfies

$$H(\{a_i\}) \le L(\{\lambda(a_i)\}) \le H(\{a_i\}) + 1;$$

$$H(\{a_i\}) = -\sum_{i=1}^{n} p(a_i)\log(p(a_i));$$

$$\sum_{i=1}^{n} p(a_i) = 1; \forall i, 0 \le p(a_i) \le 1$$

Note: Entropy is also the average number of bits required to encode the values of the random variable. **It is not possible to code the intensity values of an image with fewer bits per pixel than its entropy.**

Entropy of the random variable (set of alphabets) – measure of uncertainty of the random variable, or the measure of how much a random variable surprises you, or the average number of bits required to store a random variable.

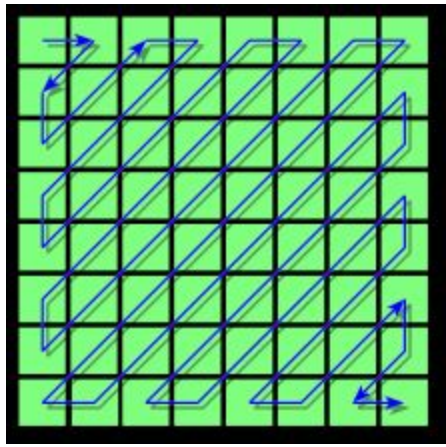
Recall: Entropy is **minimum** in the case where the random variable takes on only one value. It is the **maximum** when it can take on any one of some *k* different values, each with equal probability.

Zig-zag ordering

- The quantized DCT coefficients are arranged now in a zigzag order as follows. The zig-zag pattern leaves a **bunch of consecutive zeros at the end**.

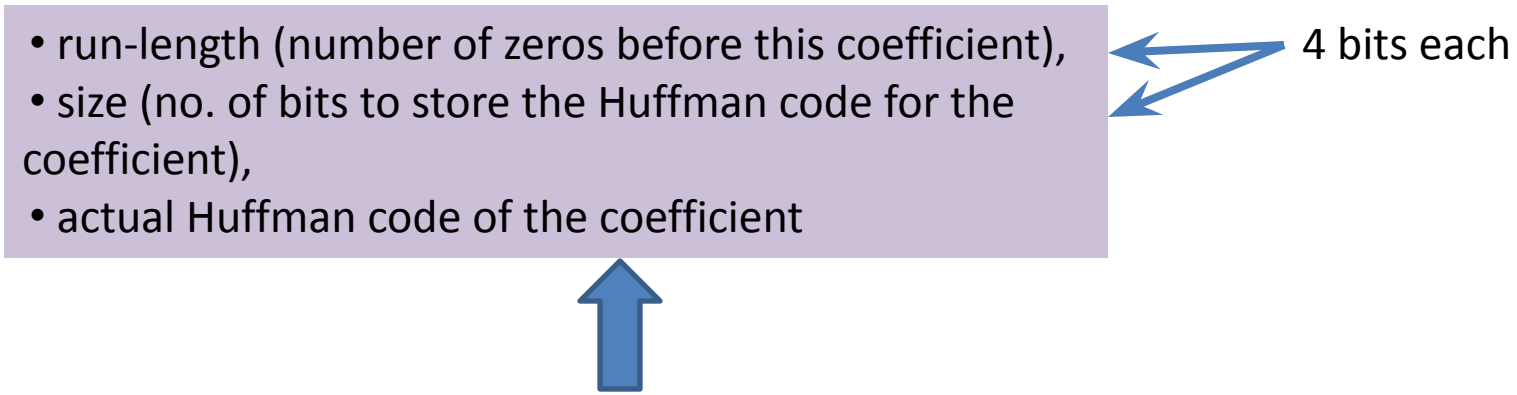
$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

−26
−3 0
−3 −2 −6
2 −4 1 −3
1 1 5 1 2
−1 1 −1 2 0 0
0 0 0 −1 −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



Run length encoding

- The non-zero re-ordered quantized DCT coefficients (except for the DC coefficient) are written down in the following format:



We refer to the above set as a **triple**. In case there are more than 15 zeros in between 2 non-zero AC coefficients, a special triple is inserted. That triple is **(15,0,0)**. If there are a large number of trailing zeros at the end of a block, we but in an **"end of block"** triple given as (0,0).

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

−26

−3 0

−3 −2 −6

2 −4 1 −3

1 1 5 1 2

−1 1 −1 2 0 0

0 0 0 −1 −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, 2)(-3); (1, 2)(-3); (0, 2)(-2); (0, 3)(-6); (0, 2)(2); (0, 2)(-4); (0, 1)(1); (0, 2)(-3); (0, 1)(1); (0, 1)(1); (0, 3)(5); (0, 1)(1); (0, 2)(2); (0, 1)(-1); (0, 1)(1); (0, 2)(2); (5, 1)(-1); (0, 1)(-1); (0, 0).

Encoding DC coefficients

- The difference between the DC coefficient of the current and previous patch is encoded and stored.
- These difference values are Huffman encoded using a separate table (different from the Huffman table used for AC coefficients).
- The DC coefficient of the first patch is stored explicitly.

JPEG encoded file

- Begins with a header that contains information such as **size of file**, whether **color or grayscale**, the table of different alphabets (i.e. DCT coefficient values and their Huffman codes) and the quantization matrix.
- This is followed by a bit stream containing triples of the form: (run-length, the length of the Huffman code of the coefficient, and the Huffman code for the coefficient).

JPEG DECODER

JPEG decoding

- Perform Huffman decoding and obtain the DCT coefficients (AC).
- Multiply the AC coefficients point-wise with the entries in the quantization matrix.
- Compute the DC coefficients for each patch using the differences between the DC coefficients of successive patches. Multiply by the appropriate entry from the quantization matrix.
- Reconstruct the image patches of size 8 x 8 using the inverse DCT. Add 128 to the intensity values in the patch.
- **Note: During JPEG encoding, the round-off errors from the quantization step can never be recovered again. Hence JPEG is overall a lossy algorithm.**

JPEG for color images

JPEG for color images

- The RGB values are converted to the YCbCr color space using:

$$Y = 16 + (65.481R + 128.553G + 24.966B)$$

$$C_B = 128 + (-37.79R - 74.203G + 112B)$$

$$C_R = 128 + (112R - 93.786G - 18.214B)$$

- Encode the Y, Cb, and Cr channels separately, using the "grayscale" JPEG algorithm on each channel. The Cb and Cr channels (the chrominance channels) are down-sampled by a factor of 2 in X and Y direction to further save storage space.
- The Y channel (luminance) is not down-sampled. This is because the human eye is much more sensitive to luminance than to chrominance information.

PCA on RGB values

- Why can you not separately compress the R,G,B images – instead of converting to Y, Cb, Cr?
- The answer lies in PCA!
- Suppose you take N color images and extract RGB values of each pixel (3 x 1 vector at each location).
- Now, suppose you build an eigenspace out of this – you get 3 eigenvectors, each corresponding to 3 different eigenvalues.

PCA on RGB values

- The eigenvectors will look typically as follows:

  0.5952 0.6619 0.4556

  0.6037 0.0059 -0.7972

  0.5303 -0.7496 0.3961

- Exact numbers are not important, but the first eigenvector is like an average of RGB. It is called as the **Luminance Channel** (Y). It is similar to the intensity in the HSI space.

PCA on RGB values

- The second eigenvector is like Y-B, and the third is like Y-G. These are called as the **Chrominance Channels**.
- The Y-Cb-Cr color space is related to this PCA-based space (though there are some details in the relative weightings of RGB to get Luminance and Chrominance – denoted by Cb and Cr).
- The values in the three channels Y, Cb and Cr are decorrelated, similar to the values projected onto the PCA-based channels.

PCA on RGB values

- Why does PCA produce decorrelated values (i.e. why are the values of the eigencoefficients decorrelated)?

- Recall: in PCA, we built the correlation matrix **C** from *N* original data-points $\{\mathbf{x}_i\}$, $1 \le i \le N$.

- Recall that $\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^t$.

- The eigencoefficients are given as $\{\boldsymbol{\alpha}_i\}$, $1 \le i \le N$ where $\mathbf{x}_i = \mathbf{V}\boldsymbol{\alpha}_i$ where $\mathbf{C} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$.

- The correlation matrix of the eigencoefficients is given by:

$$\frac{1}{N-1}\sum_{i=1}^{N} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^t = \frac{1}{N-1}\sum_{i=1}^{N} \mathbf{V}^{\mathrm{T}} \boldsymbol{x}_i \boldsymbol{x}_i^t \mathbf{V} = \mathbf{V}^{\mathrm{T}}\mathbf{C}\mathbf{V} = \boldsymbol{\Lambda}$$

PCA on RGB values

- The correlation matrix of the eigencoefficients is given by:

$$\frac{1}{N-1}\sum_{i=1}^{N}\boldsymbol{\alpha}_i\boldsymbol{\alpha}_i^t = \frac{1}{N-1}\sum_{i=1}^{N}\mathbf{V}^{\mathbf{T}}\boldsymbol{x}_i\boldsymbol{x}_i^t\mathbf{V} = \mathbf{V}^{\mathbf{T}}\mathbf{C}\mathbf{V} = \boldsymbol{\Lambda}$$

- This is a **diagonal** matrix (of eigenvalues).

- Which means that the different elements of the vector of eigencoefficients are decorrelated.

PCA on RGB values

- Why is it important to have decorrelated values in compression?
- Because if the variables were not decorrelated, any operations you perform on one variable have an effect on the other variables.
- So if you compressed the R image, G image and B image separately, a change in the R values (due to quantization) unintentionally affects the G and B values (and vice versa).
- To prevent this, you convert the color values from RGB to a decorrelated color space such as YCbCr.

PCA on RGB values

- The luminance channel (Y) carries most information from the point of view of human perception, and the human eye is less sensitive to changes in chrominance.
- This fact can be used to assign coarser quantization levels (i.e. fewer bits) for storing or transmitting Cb and Cr values as compared to the Y channel. This improves the compression rate.
- The JPEG standard for color image compression uses the YCbCr format. For an image of size M x N x 3, it stores Y with full resolution (i.e. as an M x N image), and Cb and Cr with 25% resolution, i.e. as M/2 x N/2 images.

R channel

B channel

G channel

Image containing eigencoefficient value corresponding to 1st eigenvector (with maximum eigenvalue)

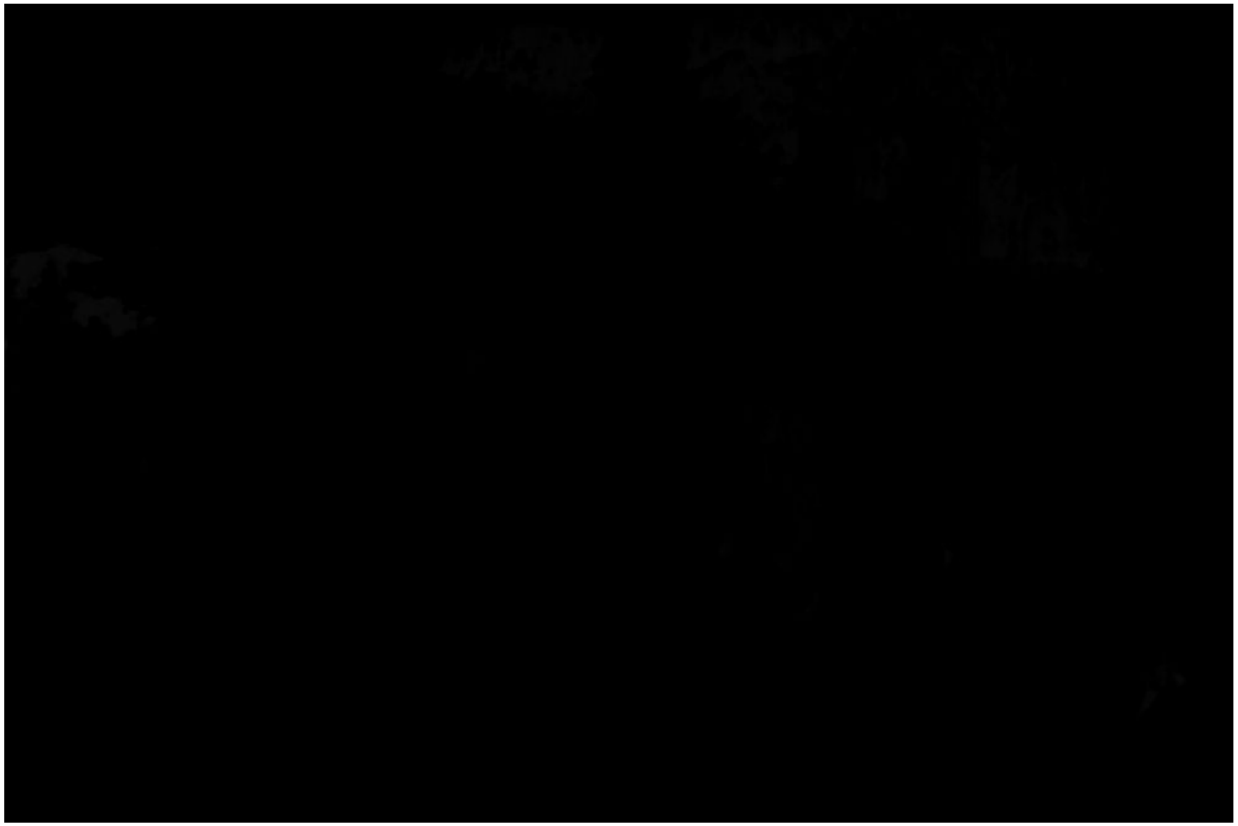
Image containing eigencoefficient value corresponding to 2nd eigenvector (with second largest eigenvalue)

Image containing eigencoefficient value corresponding to 3rd eigenvector (with least eigenvalue)

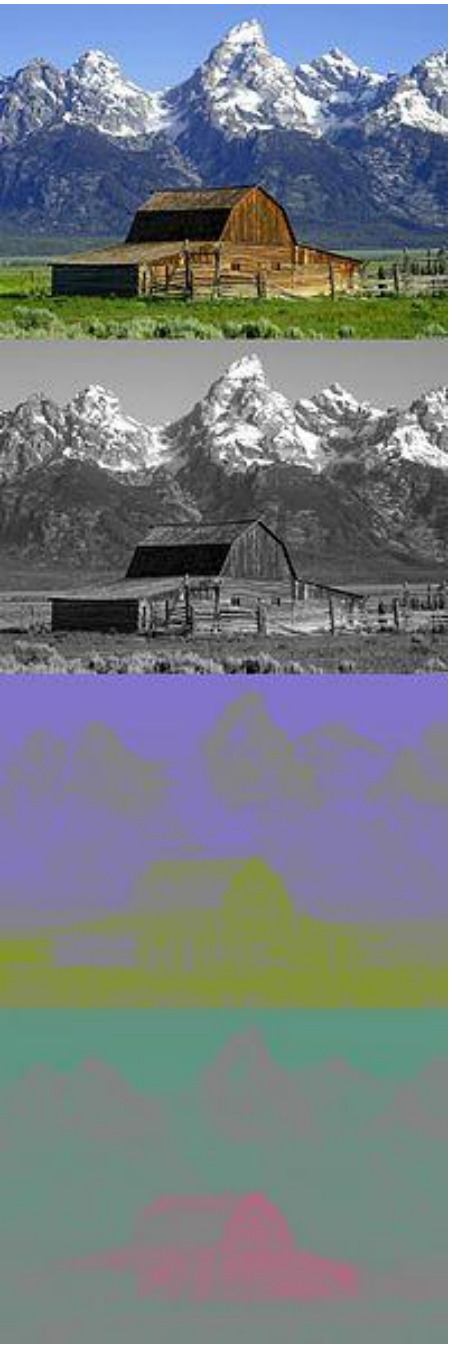
The variances of the three eigen-coefficient values:
8411, 159.1, 71.7

Y channel

Cb channel



Cr channel

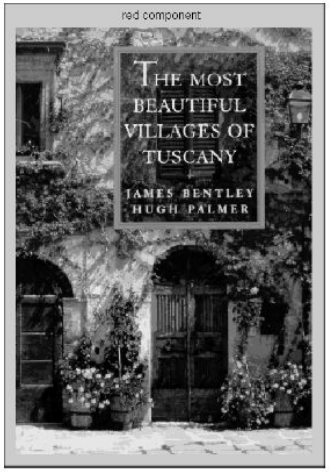


RGB and its corresponding Y, Cb, Cr channels

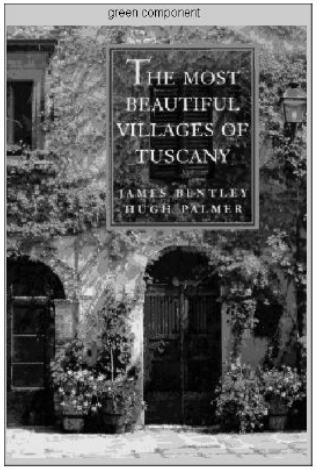
$$Y = 16 + (65.481R + 128.553G + 24.966B)$$

$$C_B = 128 + (-37.79R - 74.203G + 112B)$$

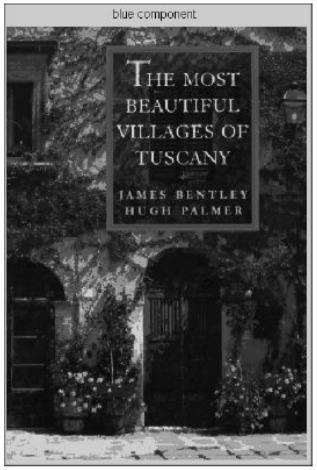
$$C_R = 128 + (112R - 93.786G - 18.214B)$$



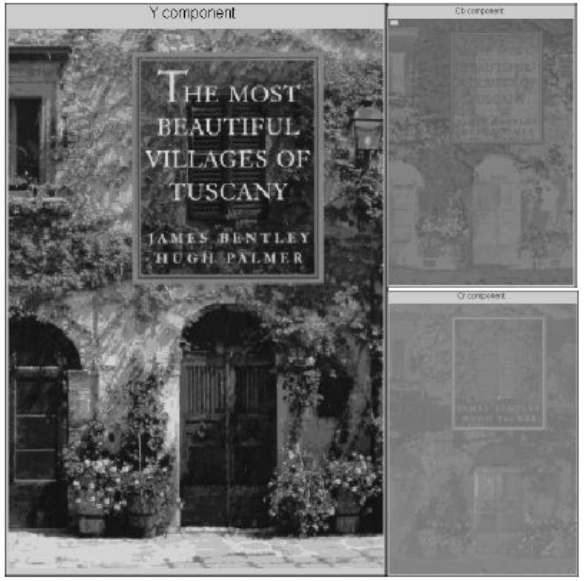
R



G



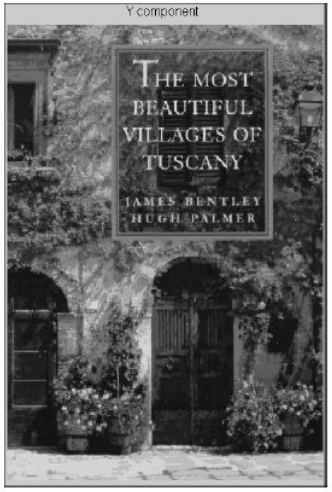
B



Cb

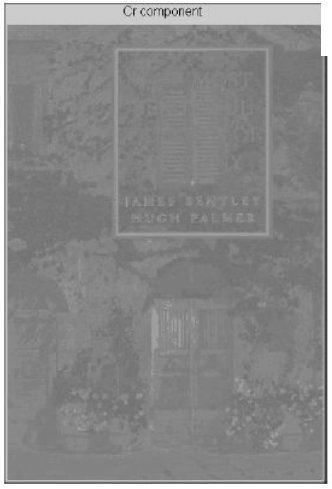
Cr

Y



Y

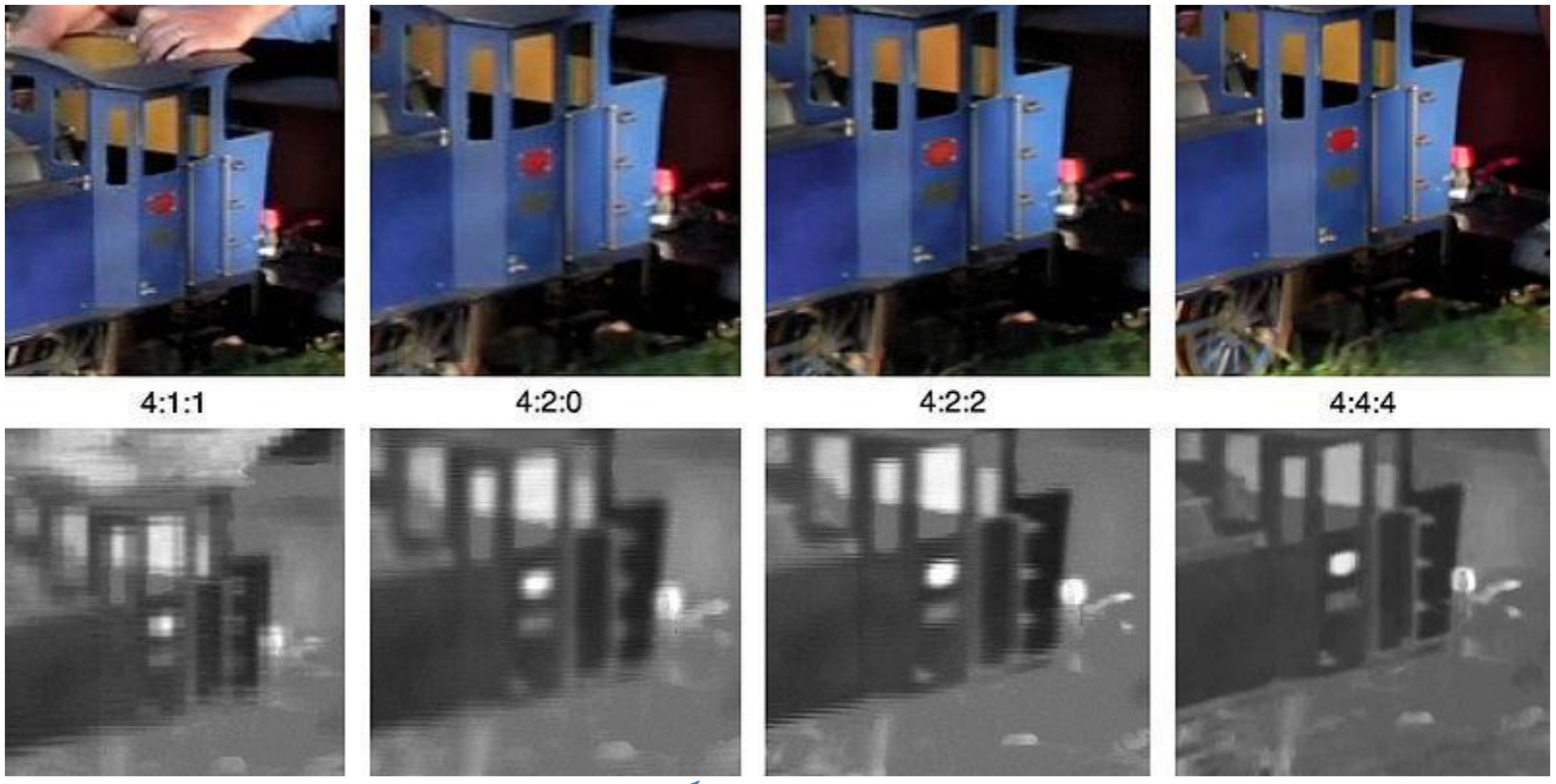
Cb



Cr

http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-11-4.pdf

Down-sampling of Cb and Cr in X and Y directions by a factor of 2	Down-sampling of Cb and Cr only in X direction by a factor of 2	No down-sampling of chrominance or luminance channels



Cb channel under different down-sampling factors

https://en.wikipedia.org/wiki/Chroma_subsampling#/media/File:Colorcomp.jpg

Modes of JPEG compression

- **Sequential:** encoding and decoding of patches takes place in left to right, top to bottom order.
- **Progressive:** encoding and decoding in multiple scans, each one with finer quantization levels.
- **Hierarchical:** encoding and decoding performed at different scales.

Sequential

Progressive

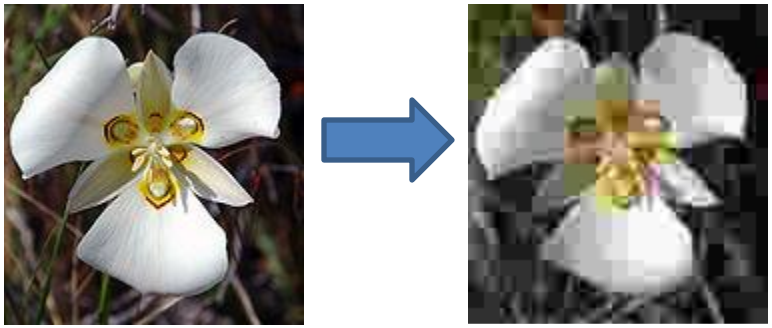
Hierarchical

Commonly seen in web applications (e.g.: Facebook)

http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-11-4.pdf

JPEG artifacts

- Seam artifacts at patch boundaries (more prominent for lower *Q* values).
- Ringing artifacts around edges.
- Some loss of edge and textural detail.
- Color artifacts



Video Compression

Need for video compression

- Huge data – typical HDTV has frames of size 1920 x 1080 and 30 fps frame-rate. That is more than 1 GB per second.
- Network channel bandwidths are limited (around 20 Mbps).
- We need large rates of compression (around 1:80)!

Motion JPEG

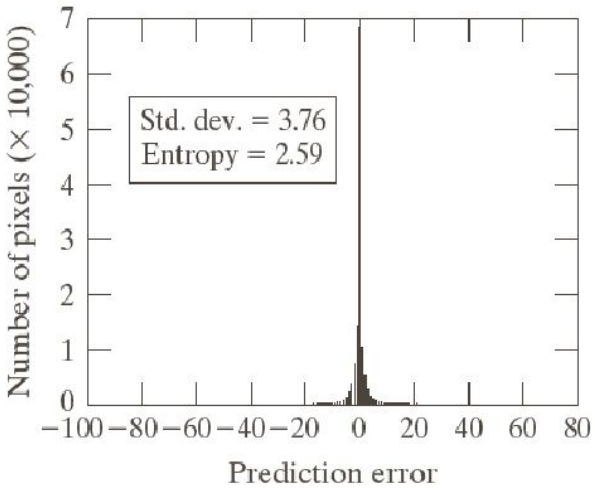
- Encode each frame using JPEG.
- That will yield only around 1:10 compression.
- This makes use of only spatial redundancy, no temporal redundancy.

MPEG (Motion Pictures Expert Group)

- Heavy use of **temporal redundancy**!
- Uses a process called **predictive coding**.
- Consider pixel *f(x,y,t)* in frame *t*. We try to predict its value, denoted as *g(x,y,t),* using a linear combinations of the values from previous *k* frames, i.e. *f(x,y,t-k)*,…,*f(x,y,t-1).*
- We simply store the error *e(x,y,t) = f(x,y,t)-g(x,y,t).*

Differential coding (First order predictive coding)

- In this method, $k = 1$, i.e. you encode the differences between consecutive frames, i.e. $e(x,y,t) = f(x,y,t) - f(x,y,t-1)$.
- For most frames, the errors are highly sparse.



a b
c d

FIGURE 8.35
(a) and (b) Two views of Earth from an orbiting space shuttle video. (c) The prediction error image resulting from Eq. (8.2-36). (d) A histogram of the prediction error. (Original images courtesy of NASA.)

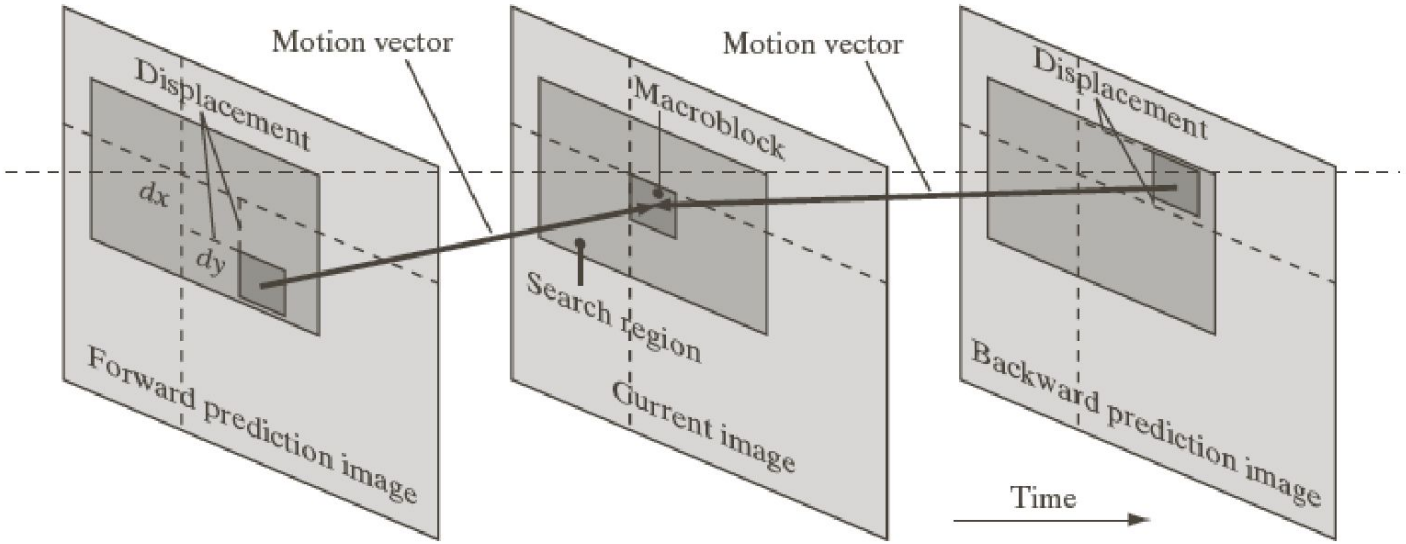
Errors are not always sparse ☹️

- Exceptions: (1) camera zoom-in and zoom-out, (2) sudden changes in viewpoint or scene content, or fade-in/fade-out effects. **In such cases, the errors will be large!**

Motion compensation

- For each **macro-block** (typically 16 x 16 or 8 x 8 in size) in the frame to be encoded, find the most similar macro-block in a **reference frame** (which could be the previous frame, but not necessarily so).
- The difference between the pixel locations of the top-left corner of the two macro-blocks is called the **motion vector**.
- The motion vector is considered to be **constant within a macro-block**.
- Search for similar macro-blocks is restricted to a small **search window** around the original macro-block.
- The search window is **rectangular** – broader than taller (why?).

FIGURE 8.36
Macroblock motion specification.



Motion Compensation

- For many macro-blocks, the motion vectors will be 0.
- The search for similar macro-blocks is performed at a **sub-pixel** level (1/2 pixel or ¼ pixel) for more accuracy. In this case, image intensity values need to be interpolated.
- Macro-block **similarity measure** is one of the following:

$$SE(x,y) = \sum_{i=1}^{m}\sum_{j=1}^{n}(f(x+i,y+j,t) - f(x+i+dx,y+j+dy,t-1))^2$$

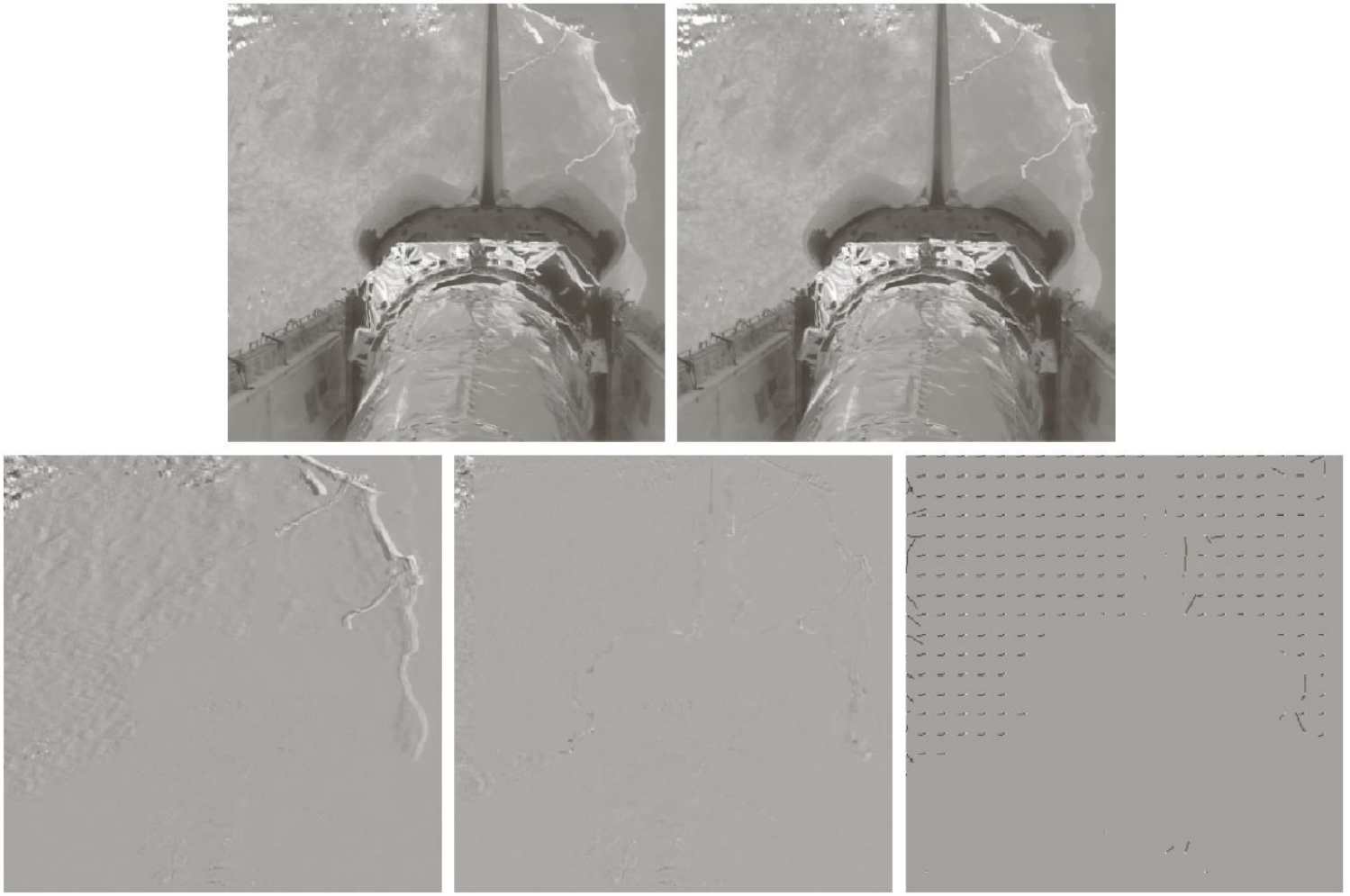
$$AE(x,y) = \sum_{i=1}^{m}\sum_{j=1}^{n}|f(x+i,y+j,t) - f(x+i+dx,y+j+dy,t-1)|$$

$$8 \le dx \le 64, 8 \le dy \le 64$$

In color videos, only luminance (i.e. Y) channel is used in similarity measure

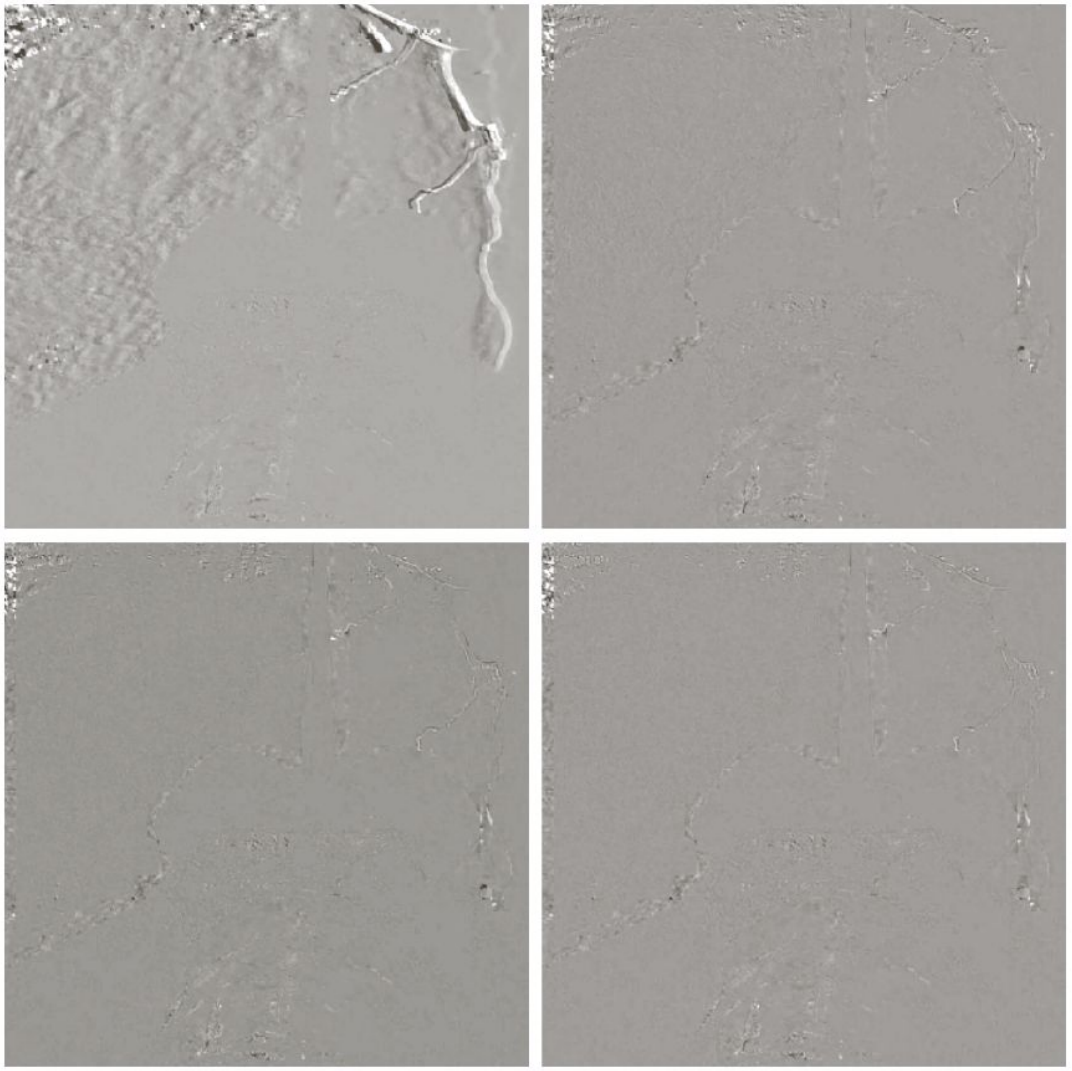
Motion compensation

- Many a time, there are more than one similar block. In such cases, the **spatially closest** block is chosen.
- Note: we are not concerned with the accuracy of the motion estimate. It is only a means towards the larger goal – of compression.
- The inter-frame differences are computed **after motion compensation**. This hugely improves the sparsity of these differences.
- In other words, don't blindly compute the difference between macro-blocks at corresponding locations in frames *t* and *t-1*. Compute the difference between the current macro-block in frame *t* and its most similar match in frame *t-1.*



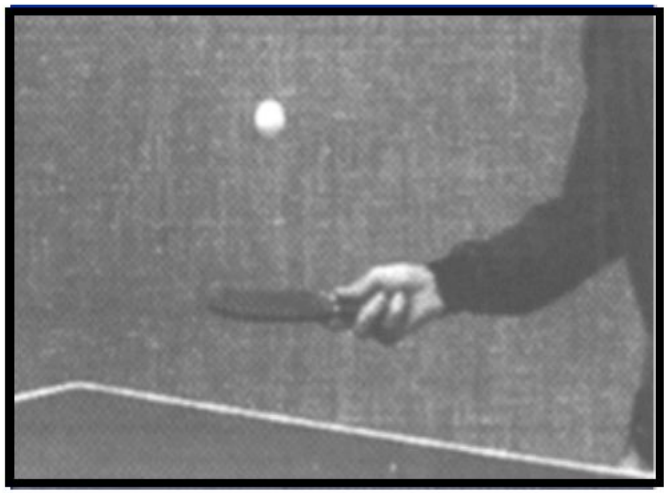
a b
c d e

FIGURE 8.37 (a) and (b) Two views of Earth that are thirteen frames apart in an orbiting space shuttle video. (c) A prediction error image without motion compensation. (d) The prediction residual with motion compensation. (e) The motion vectors associated with (d). The white dots in (d) represent the arrow heads of the motion vectors that are depicted. (Original images courtesy of NASA.)

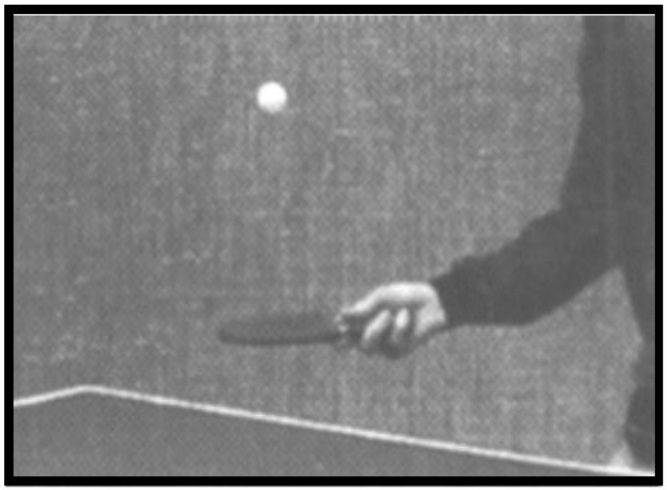


a b
c d

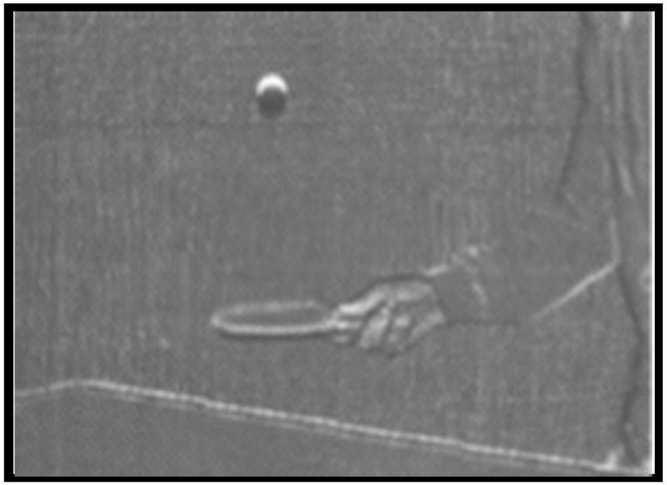
FIGURE 8.38
Sub-pixel motion compensated prediction residuals: (a) without motion compensation; (b) single pixel precision; (c) $\frac{1}{2}$ pixel precision; and (d) $\frac{1}{4}$ pixel precision. (All prediction errors have been scaled to the full intensity range and then multiplied by 2 to increase their visibility.)



Frame N



Frame N+1



Difference frame without motion prediction

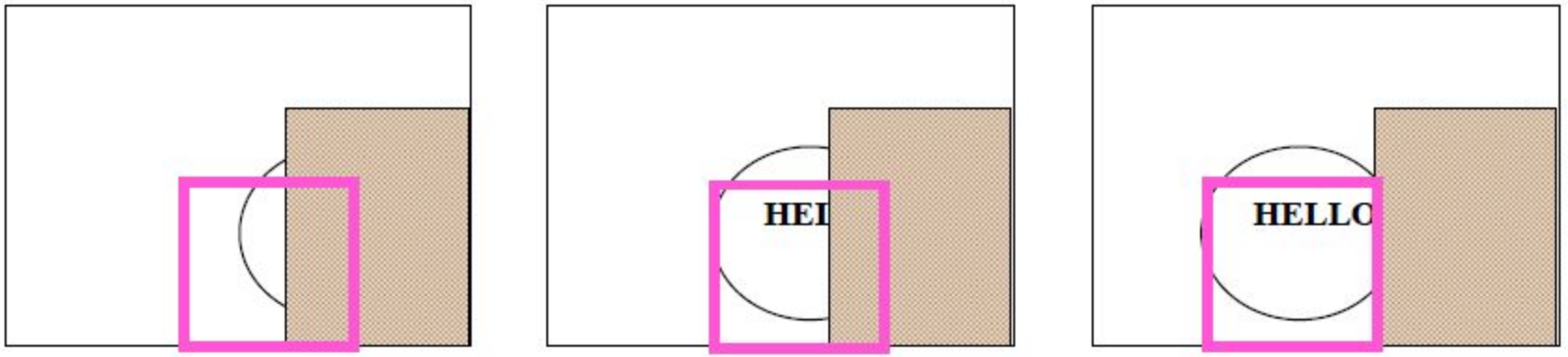
Difference frame with motion prediction

http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-11-5.pdf

Motion Compensation

- For each **macro-block** (typically 16 x 16 or 8 x 8 in size) in the frame to be encoded, find the most similar macro-block in a **reference frame**.
- If the reference frame is the previous one or the previous reference frame (see two slides later for more details), the current frame (the one being encoded) is called the **P-frame**.
- If the reference frame is a combination of the previous frame and the next one, the current frame is called the **B-frame**.
- A B-frame can use two motion vectors – one for previous and one for next frame.

Example: why do we need B-frames?



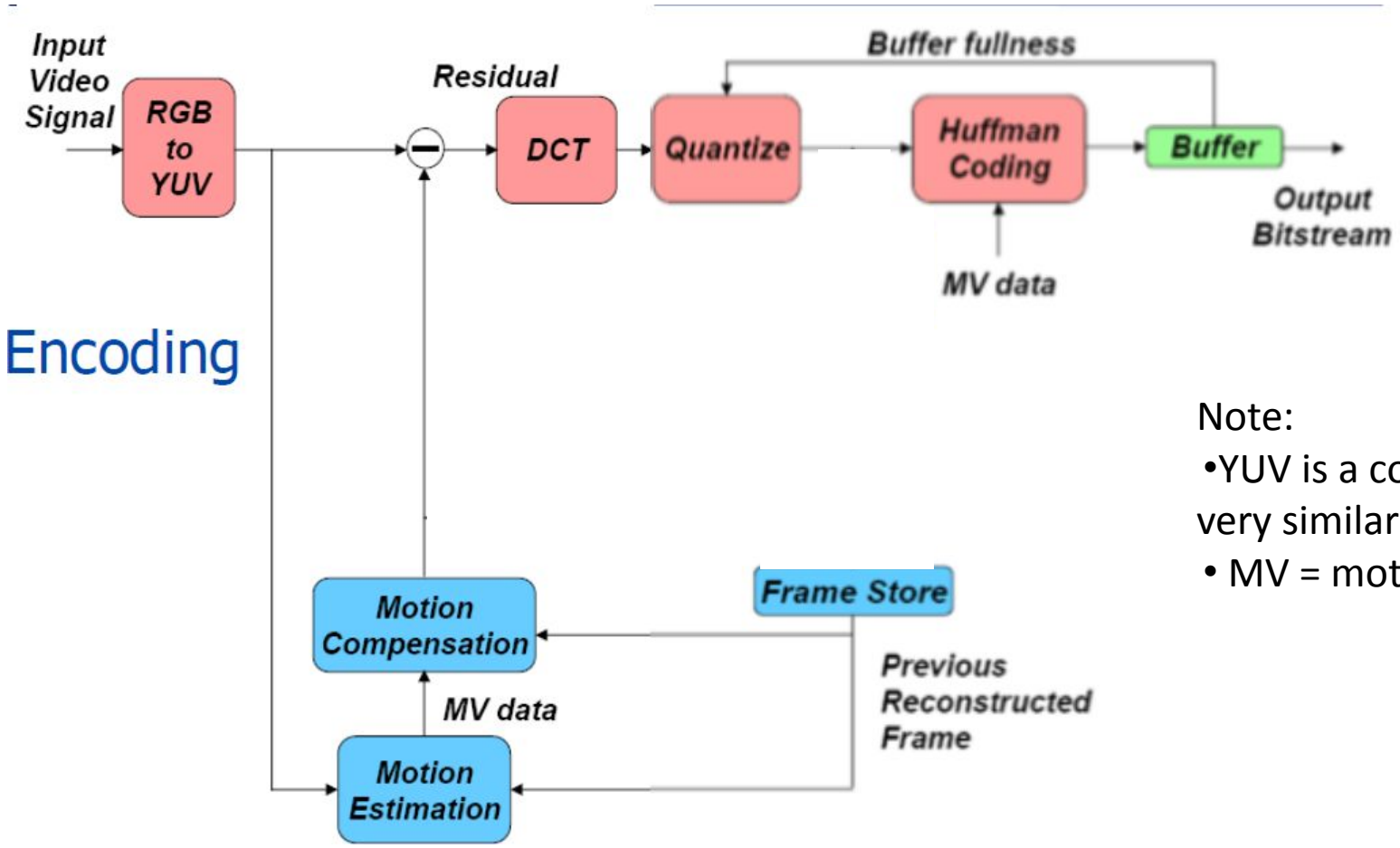
Frame N-1 Frame N Frame N+1

http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-11-5.pdf

No Motion Compensation here!

- For some frames that rest on shot-boundaries (i.e. sudden changes in content), there is no advantage to performing motion compensation.
- Such frames are called **I-frames (independent frames)**. These usually act as reference frames for other frames to be differentially encoded.
- I-frames can be detected by the presence of very frequently low similarity values during search for macro-blocks.

MPEG encoder



Note:

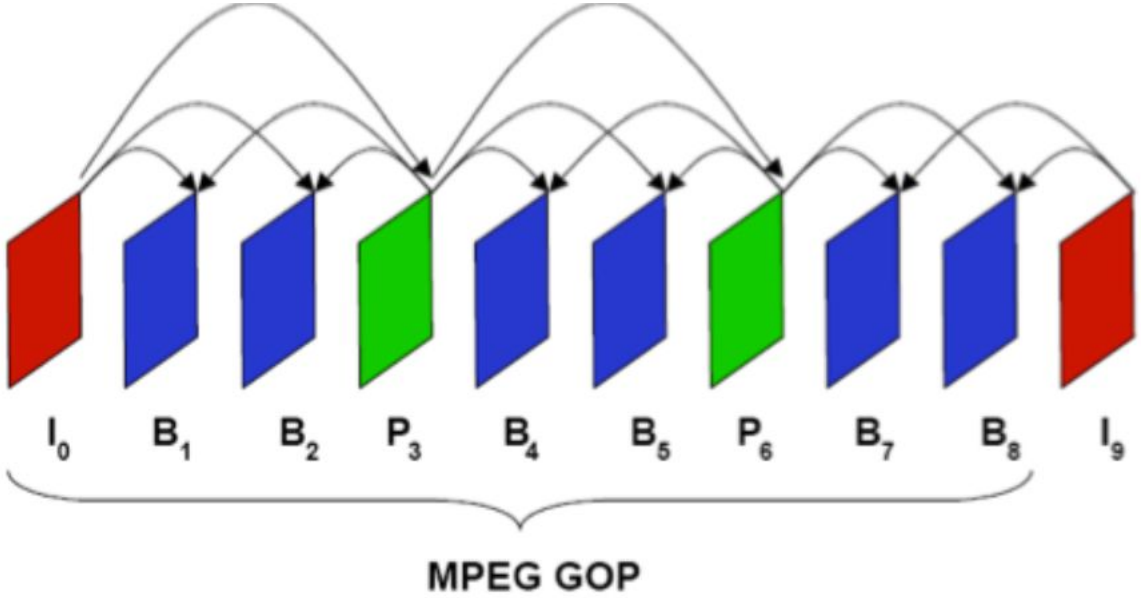
- YUV is a color-space very similar to YCbCr.
- MV = motion vector

http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-11-5.pdf

MPEG encoder

- The I-frames are encoded using JPEG.
- The B-frames and P-frames are also encoded using JPEG, but the DCT is computed on macro-blocks from the motion-compensated residual image as follows:
  - We have already computed motion vectors w.r.t. the reference frame.
  - Compute a residual image by calculating differences between macro-blocks in the current frame and the most-similar matching macro-blocks (as given by the motion vector) from the reference image. This is called motion-compensated frame differencing.
  - **Note: the motion vector for the macro-block also needs to be stored.** For this, motion-vectors from several macro-blocks are collected together and Huffman-encoded. Only the non-zero motion vectors are encoded.

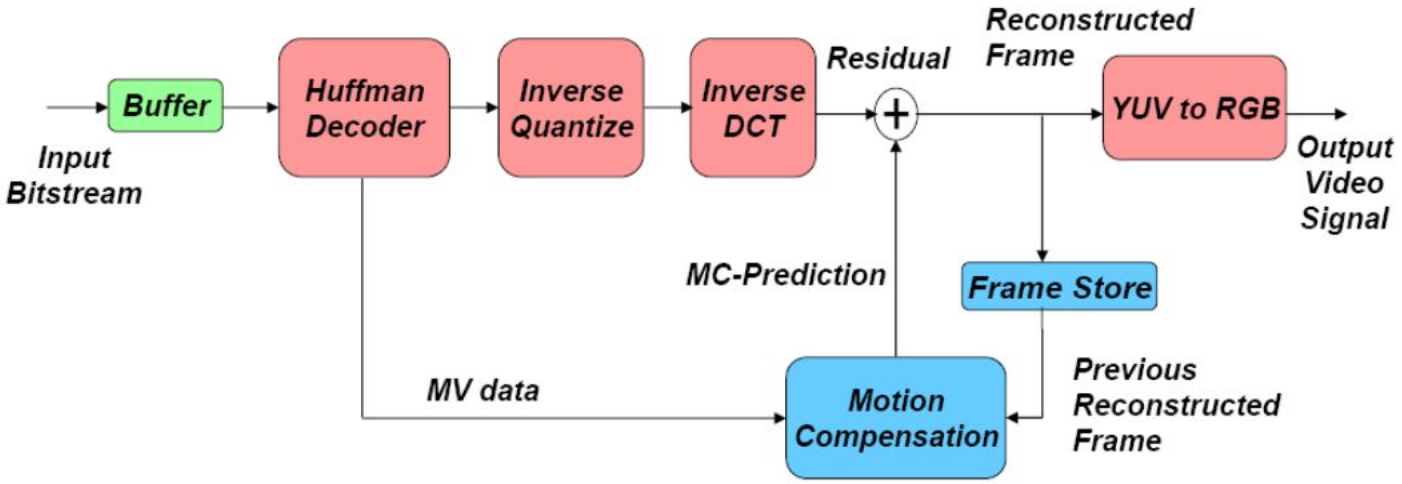
Display Order and Transmission Order (or order in which frames are compressed) may be different!



Display order: I_0, B_1, B_2, P_3, B_4, B_5, P_6, B_7, B_8, I_9
Transmission Order: I_0, P_3, B_1, B_2, P_6, B_4, B_5, I_9, B_7, B_8

http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-15-5.pdf

MPEG decoder



Note:

- YUV is a color-space very similar to YCbCr.