

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.
- 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).

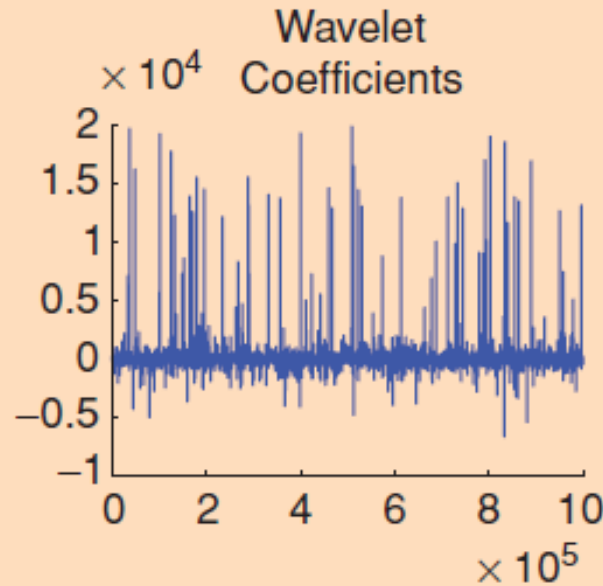
Lossy image compression

- 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



(a)



(b)



(c)

[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. As we describe in "Undersampling and Sparse Signal Recovery," this image can be perfectly recovered from just 96,000 incoherent measurements.

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

- A user-specified 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:

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

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

$Q = 100$,
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 = $1/23$

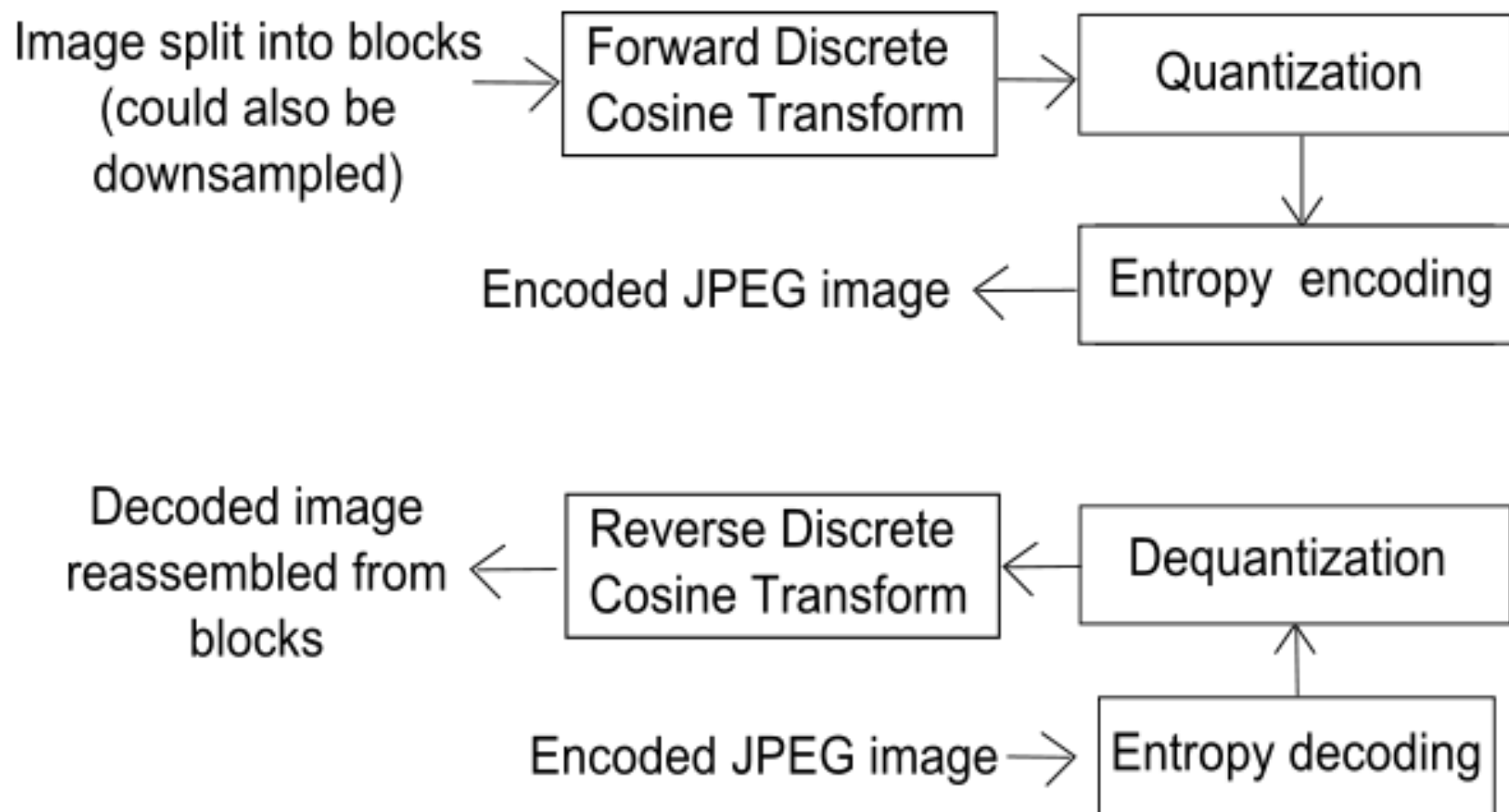


<http://en.wikipedia.org/wiki/JPEG>

Steps of the JPEG algorithm: 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.



STEP 1: Discrete Cosine Transform (DCT)

Discrete Cosine Transform (DCT) in 1D

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

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

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

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

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

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

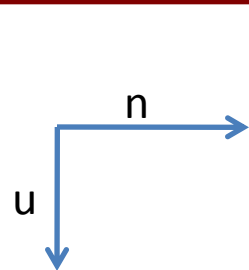
$$\tilde{a}_N^{un} = a_N^{*un} \text{ (complex conjugate)}$$

Discrete Cosine Transform (DCT) in 1D

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

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

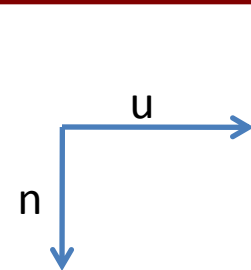
↓



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

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

↓



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

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

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

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

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)

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

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

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

- 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**.

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 in this case will have size $MN \times MN$, and it will be the Kronecker product of two DCT matrices – one of size $M \times M$, the other of size $N \times N$. 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 :

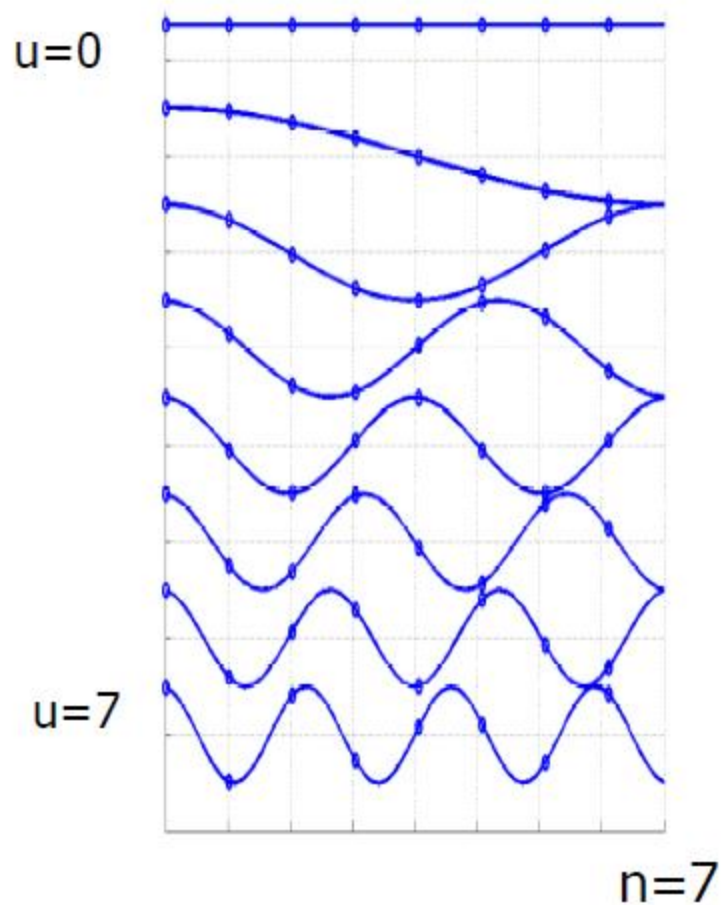
$$a_{NM}^{unvm} = \alpha(u) \alpha(v) \cos\left(\frac{\pi(2n+1)u}{2N}\right) \cos\left(\frac{\pi(2m+1)v}{2M}\right), u = 0 \dots N-1, v = 0 \dots M-1$$

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

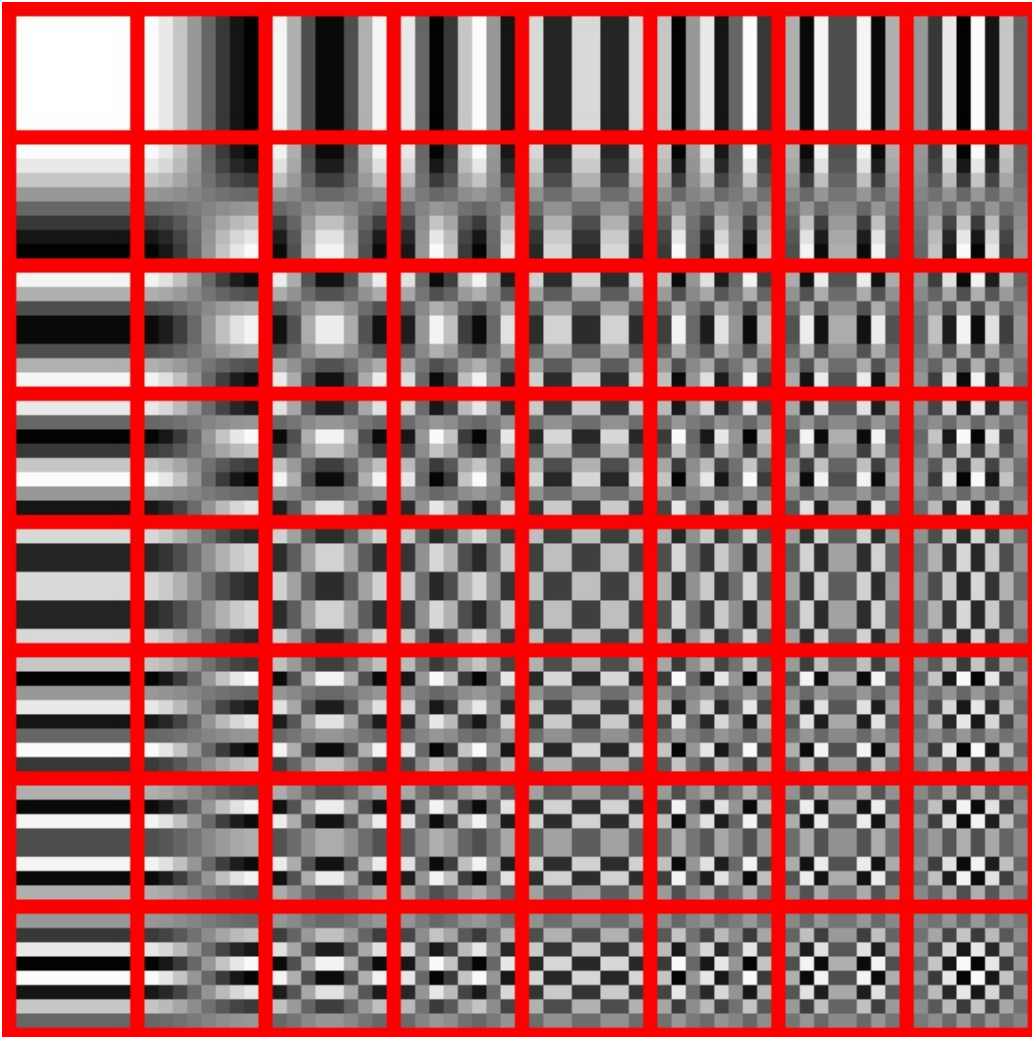
$$\alpha(v) = \sqrt{1/M} \ (v = 0), \text{ else } \alpha(v) = \sqrt{2/M}$$

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

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



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

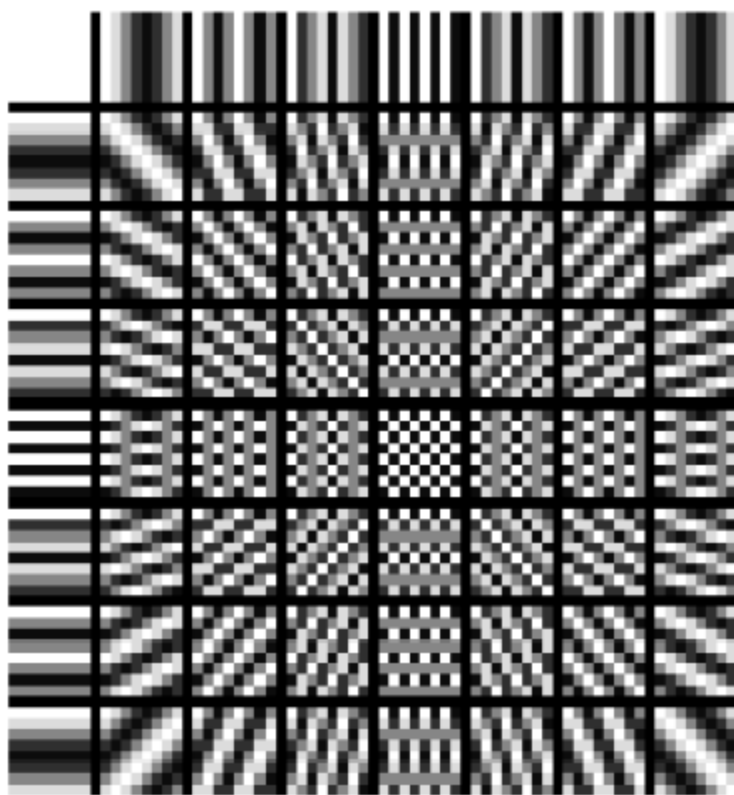


The DCT transforms an 8×8 block of input values to a [linear combination](#) of these 64 patterns. The patterns are referred to as the two-dimensional DCT *basis functions*, and the output values are referred to as *transform coefficients*.

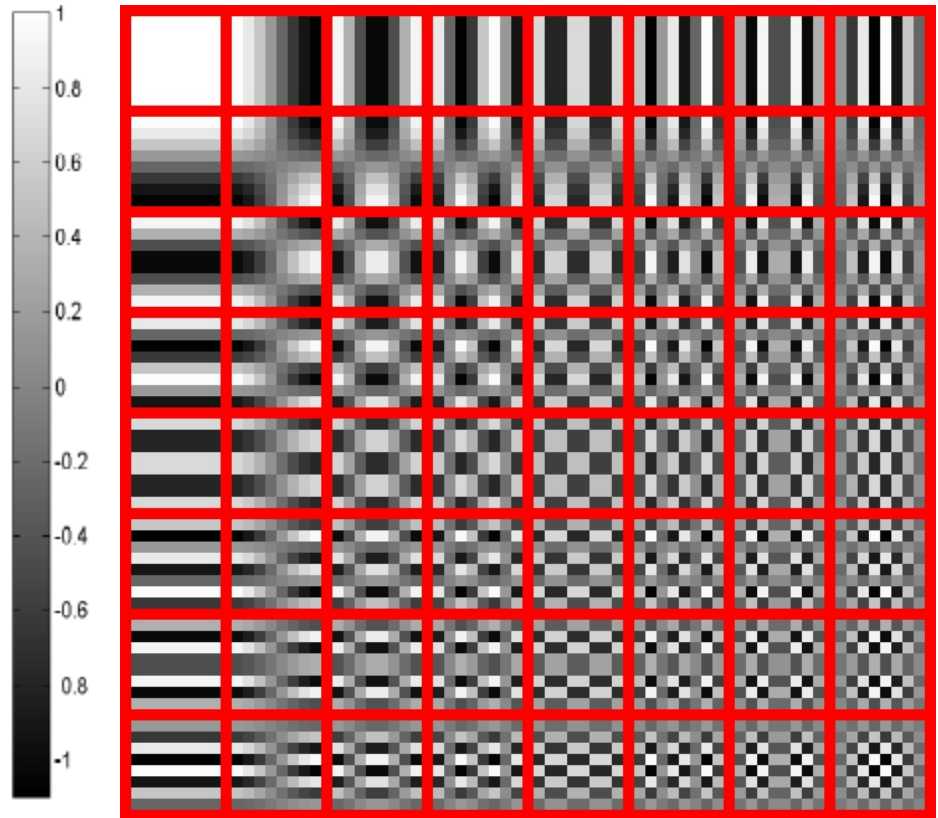
Each image here is obtained from the 8×8 outer product of a pair of DCT basis vectors. Each image is stretched between 0 and 255 – on a common scale.

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

Again: DCT is NOT the real part of the DFT



Real part of DFT



DCT

DCT on grayscale image patches

- The DCT coefficients of image patches have an amazing property. 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.

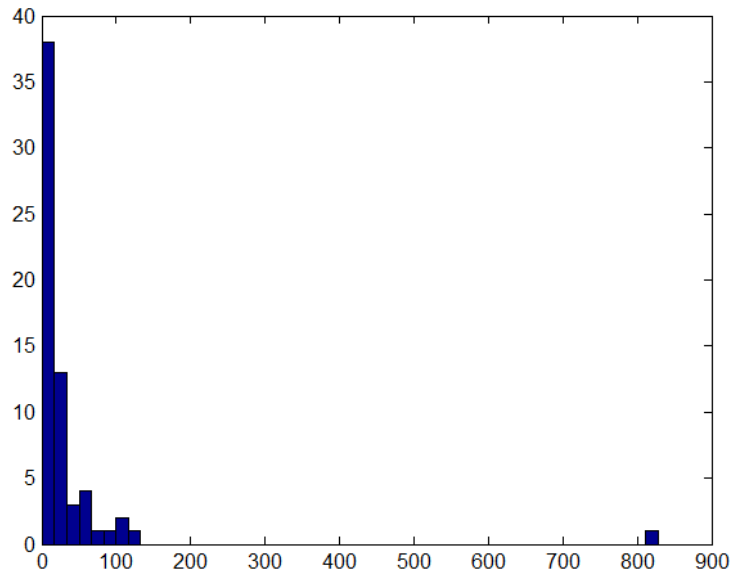
149	74	92	74	74	74	149	162
87	74	117	30	74	105	180	130
30	117	105	43	105	130	149	105
74	162	105	74	105	117	105	105
117	149	74	117	74	105	74	149
149	87	74	87	74	74	117	180
105	74	105	43	61	117	180	149
74	74	105	74	105	130	149	105



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 8×8 patches with absolute value less than 10, and then computing inverse DCT



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

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

Number of DCT coefficients of non-overlapping 8×8 patches with absolute value less than 10 was 51,045 out of a total of 65536 (64 coefficients for each 8×8 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...

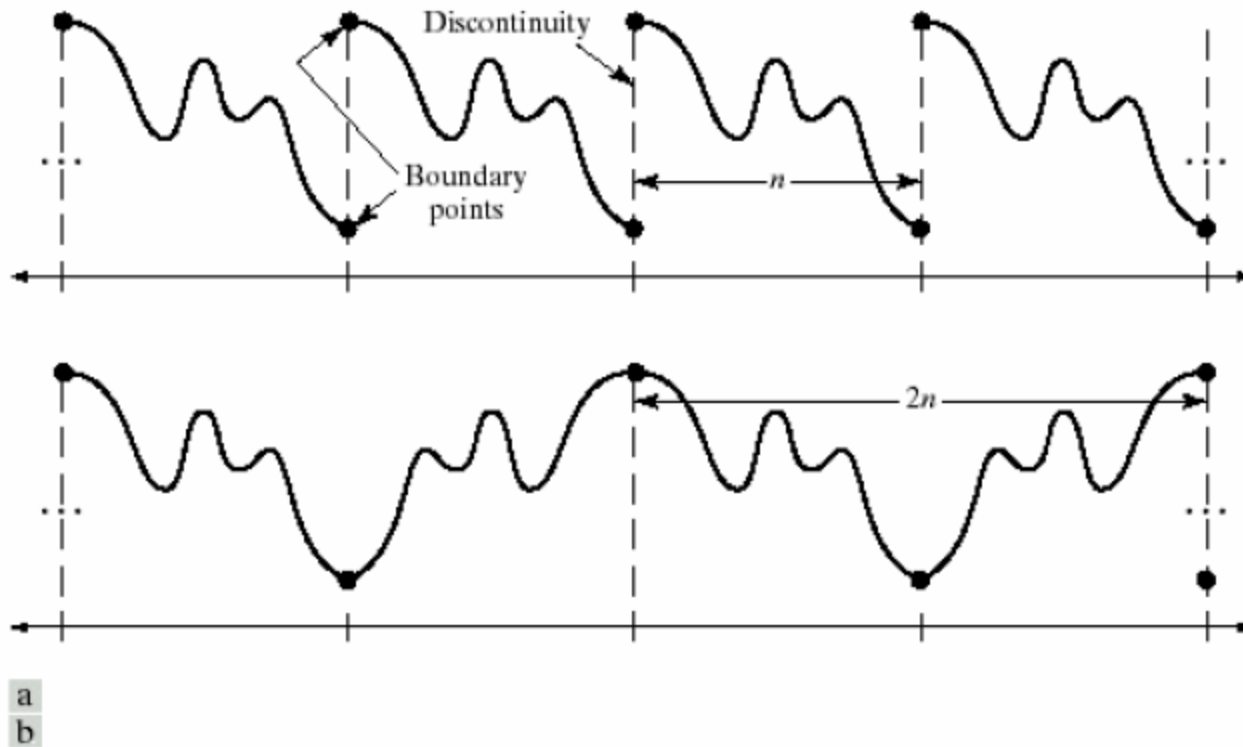
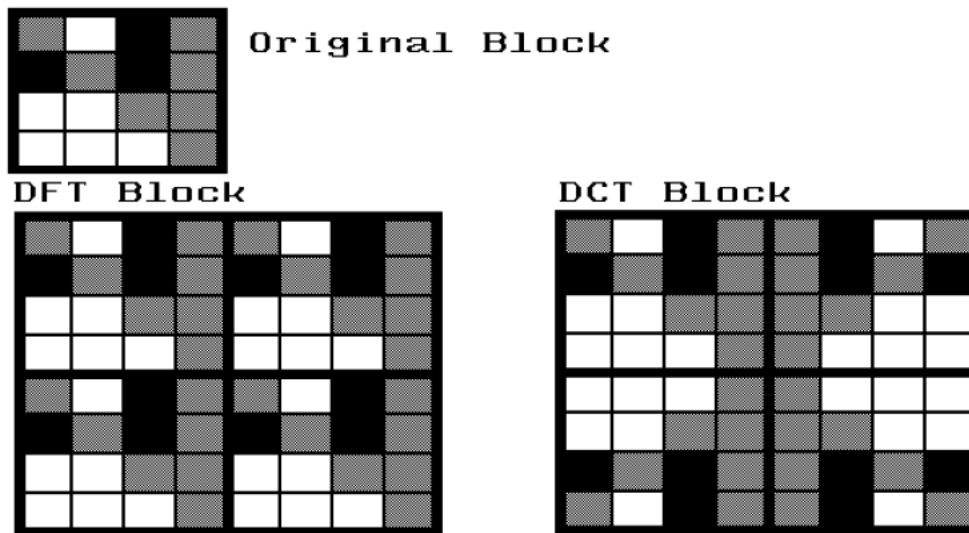


FIGURE 8.32 The periodicity implicit in the 1-D (a) DFT and (b) DCT.

In computing the DFT of a signal, 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. As against this, the discontinuities are reduced in a DCT because a reflected copy of the signal is appended to it ($2n$ -point periodicity).

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.

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

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

Reflected version of **f**, appended to **f**.

It can be shown that :

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

where

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

$$= \sqrt{2/N} / 2, \text{ for other values of } u$$

- Note: u is frequency

DCT of **f** is computed from the DFT of the sequence of double length as **f**. Only the first N frequencies are picked, i.e. u lies from 0 to $N-1$.

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

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

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

Replace n by $2N-n-1$

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

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

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

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

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

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

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

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.

```
clear;
clc;

N = 17;
f = rand(1,N);
ff = [f f(end:-1:1)];

fprintf('\nDCT of f using dct command of MATLAB: ');
df = dct(f);
display(df);

fprintf('\nDCT of f derived from fft: ');
facs = zeros(1,2*N); facs(2:2*N) = sqrt(2/N); facs(1) = sqrt(1/N);
df2 = fft(ff).*exp(-1i*pi*(0:2*N-1)/(2*N)).*facs/2;
% the second term above is  $\exp(-j \pi u / N)$  from the previous slide.
% facs refers to the constant factors with which you must multiply the
% values to get the conventional DCT and then divide by 2
df2 = df2(1:N); df2 = real(df2);
% df2 is in theory real, but due to numerical issues, insignificant complex
% parts arise.
display(df2);
```

Code to verify the previous relationship. Try it out!

http://www.cse.iitb.ac.in/~ajitvr/CS663_Fall2014/dct_dft_relation.m

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:

$\tilde{\mathbf{q}}_i^{(k)} = \mathbf{U}\tilde{\boldsymbol{\theta}}_i$, where $\tilde{\boldsymbol{\theta}}_i$ is obtained by setting to 0 all except k coefficients (the same k coefficients are retained for all patches)

Which is the best orthonormal basis?

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

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

Which is the best orthonormal basis?

- The answer is the PCA basis, i.e. the set of k eigenvectors of the correlation matrix \mathbf{C} , corresponding to the k largest eigen-values. Here is \mathbf{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:

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

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

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

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

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

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.

Experiment

- Suppose you extract $M \sim 100,000$ small-sized (8×8) 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 from the 2D DCT matrix (Kronecker product of 2 1D DCT matrices).

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

V_C : Eigenvectors of column-
column correlation matrix

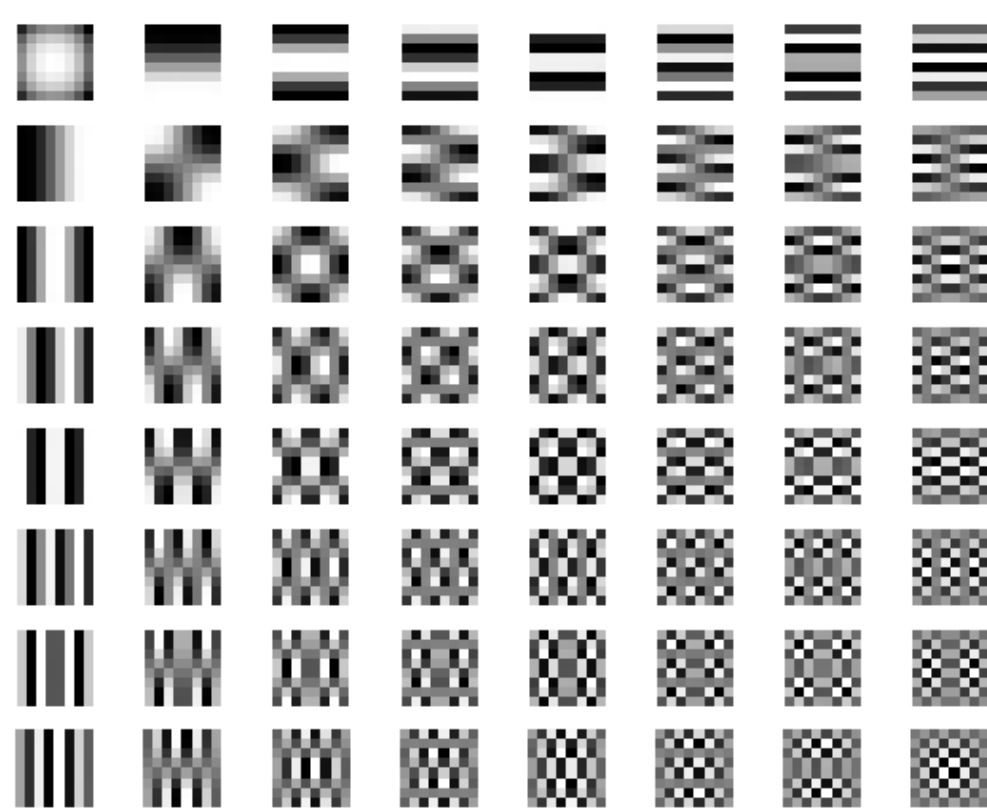
0.3520	-0.4461	-0.4305	0.4224	0.3696	0.3247	0.2342	0.1283
0.3537	-0.4338	-0.2345	-0.0114	-0.3000	-0.4671	-0.4814	-0.3028
0.3545	-0.3086	0.1662	-0.4896	-0.4007	0.0359	0.4261	0.4102
0.3548	-0.1145	0.4763	-0.3031	0.3339	0.4198	-0.1800	-0.4713
0.3548	0.1056	0.4839	0.2926	0.3349	-0.4194	-0.1766	0.4733
0.3543	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

V_R : Eigenvectors of row-row
correlation matrix

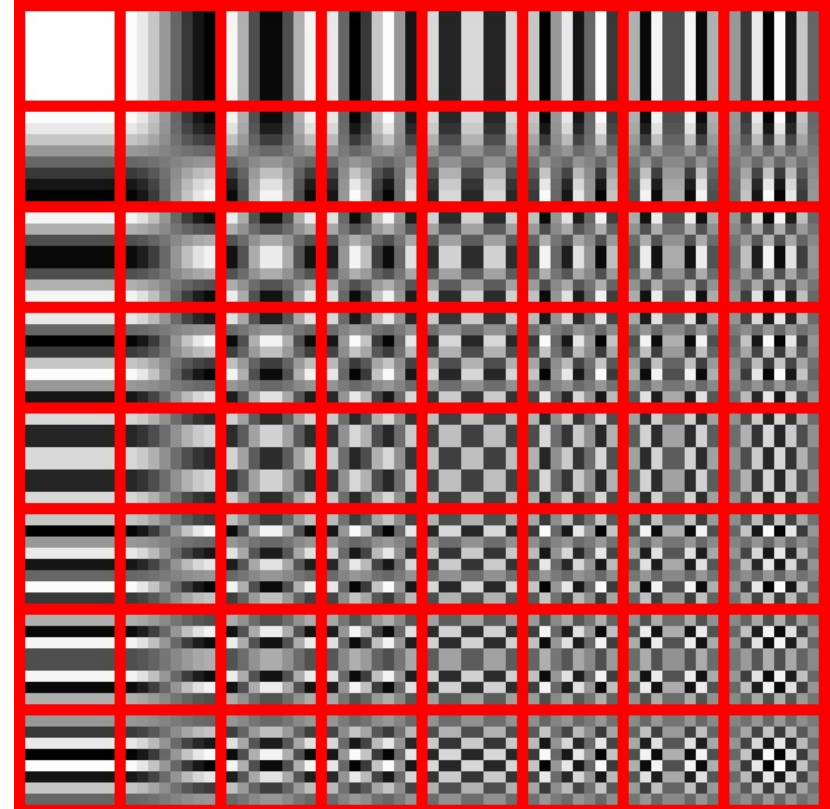
Absolute value of dot products between the columns of DCT matrix and columns of V_R (left) and V_C (right)

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 \mathbf{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 \mathbf{V} . Again, \mathbf{V} is the Kronecker product of \mathbf{V}_R and \mathbf{V}_C .



DCT bases

DCT and PCA

- DCT basis is very close to PCA basis when the data-points come from what is called as a **stationary first order Markov process** when ρ (defined below) is close to 1, i.e.

$$P(q_i | q_{i-1}, q_{i-2}, \dots, q_{i-n}) = P(q_i | q_{i-1}) \longrightarrow \text{Defn. of 1}^{\text{st}} \text{ order Markov process}$$

$$\therefore E(q_i | q_{i-1}, q_{i-2}, \dots, q_{i-n}) = E(q_i | q_{i-1})$$

$$E(q_i q_{i-1}) = \rho, E(q_i q_{i-2}) = \rho^2, \dots, E(q_i q_{i-n+1}) = \rho^{n-1}, |\rho| < 1$$

$$C = \begin{pmatrix} 1 & \rho & \rho^2 & \cdot & \rho^{n-1} \\ \rho & 1 & \cdot & \cdot & \cdot \\ \rho^2 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \rho^{n-1} & \cdot & \cdot & \cdot & 1 \end{pmatrix}, C_{ij} = E(q_i q_j)$$

For this part, you may refer to section 2.9 (equations 2.67 and 2.68) of the book “Fundamentals of Digital Image Processing” by Anil Jain. Also read, section 5.6 (equations 5.95 and 5.96).

DCT and PCA

- One can show that the eigenvectors of the covariance matrix of the form seen on the previous slide are very close to 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.

More results from the previous experiment.

See code: http://www.cse.iitb.ac.in/~ajitvr/CS663_Fall2014/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 $p \sim 0.99$

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 $p \sim 0.9888$

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 coefficients by a quantization matrix and rounding off to the nearest integer.
- This is the **lossy** part of JPEG!
- The quantization matrix on the next slide is for quality factor $Q = 50$.
- Matrices for other Q values are obtained by scaling the $Q = 50$ matrix with a constant $50/Q$.
- This effectively increases the values of the matrix for lower values of Q , and reduces the values of the matrix for higher values of Q .

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

$$M = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \cdot \quad B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot$$

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

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

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?

- Picked from psychophysical studies having the goal of trying to maximize the amount of compression – but also trying to minimize perceptually significant errors.

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:

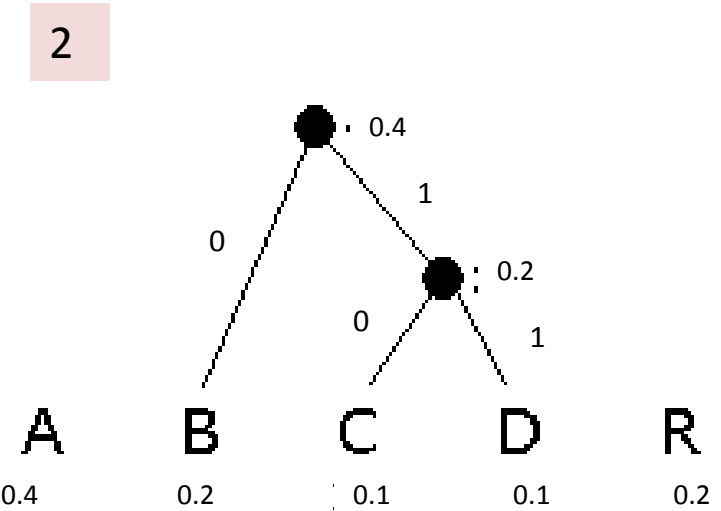
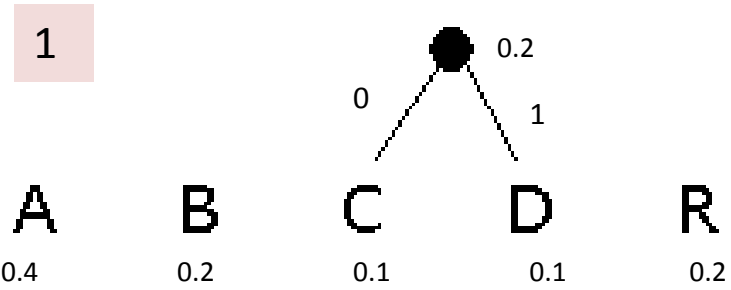
Given a set of n alphabets $\mathbf{A} = \{a_i\}$ with corresponding frequencies $\{p(a_i)\}$ (each frequency lies from 0 to 1), what prefix-free encoding yields the least average bit length? That is, which set of code-words $\{\lambda(a_i)\}$ will minimize

$$L(\{\lambda(a_i)\}) = \sum_{i=1}^n p(a_i) \boxed{|\lambda(a_i)|} \quad \leftarrow \text{Length of the code-word } \lambda(a_i)$$

Algorithm

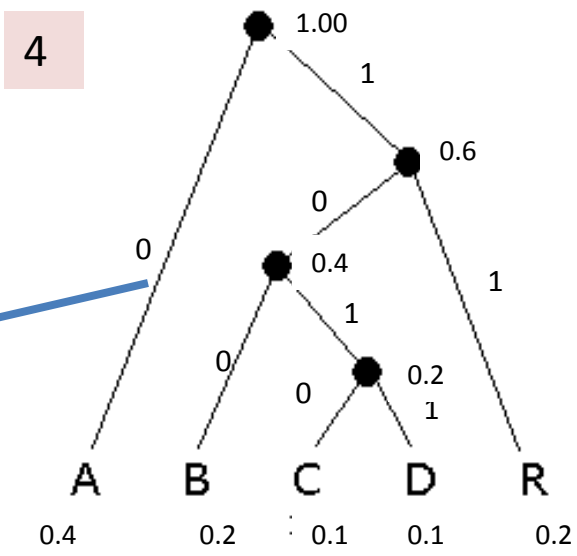
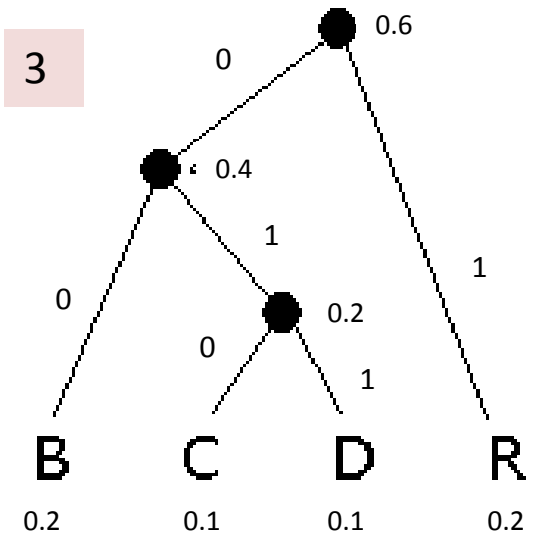
1. Sort alphabets in **increasing** order of frequency. Create a leaf node from each alphabet. These leaf nodes will belong to a binary tree called the **Huffman tree**.
2. Combine the two **lowest** frequency nodes s_1 and s_2 to create a parent node s_{12} . s_1 and s_2 will be the left and right child of s_{12} . The frequency of s_{12} is given by $p(s_{12}) = p(s_1) + p(s_2)$.
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 $p(s_{12})$.
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.

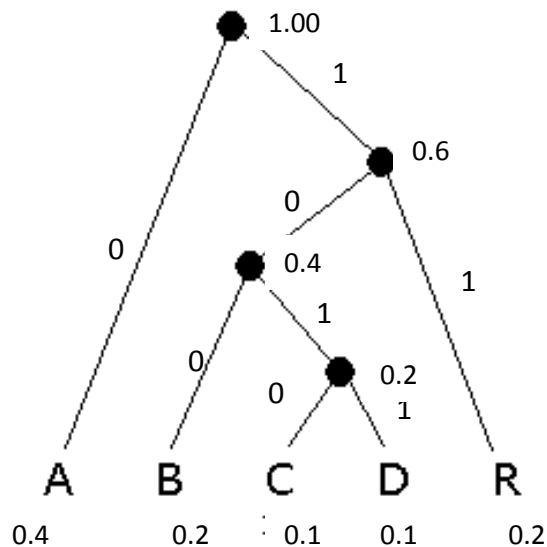
Example



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





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.

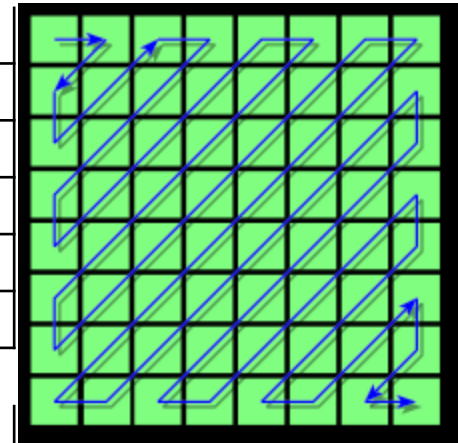
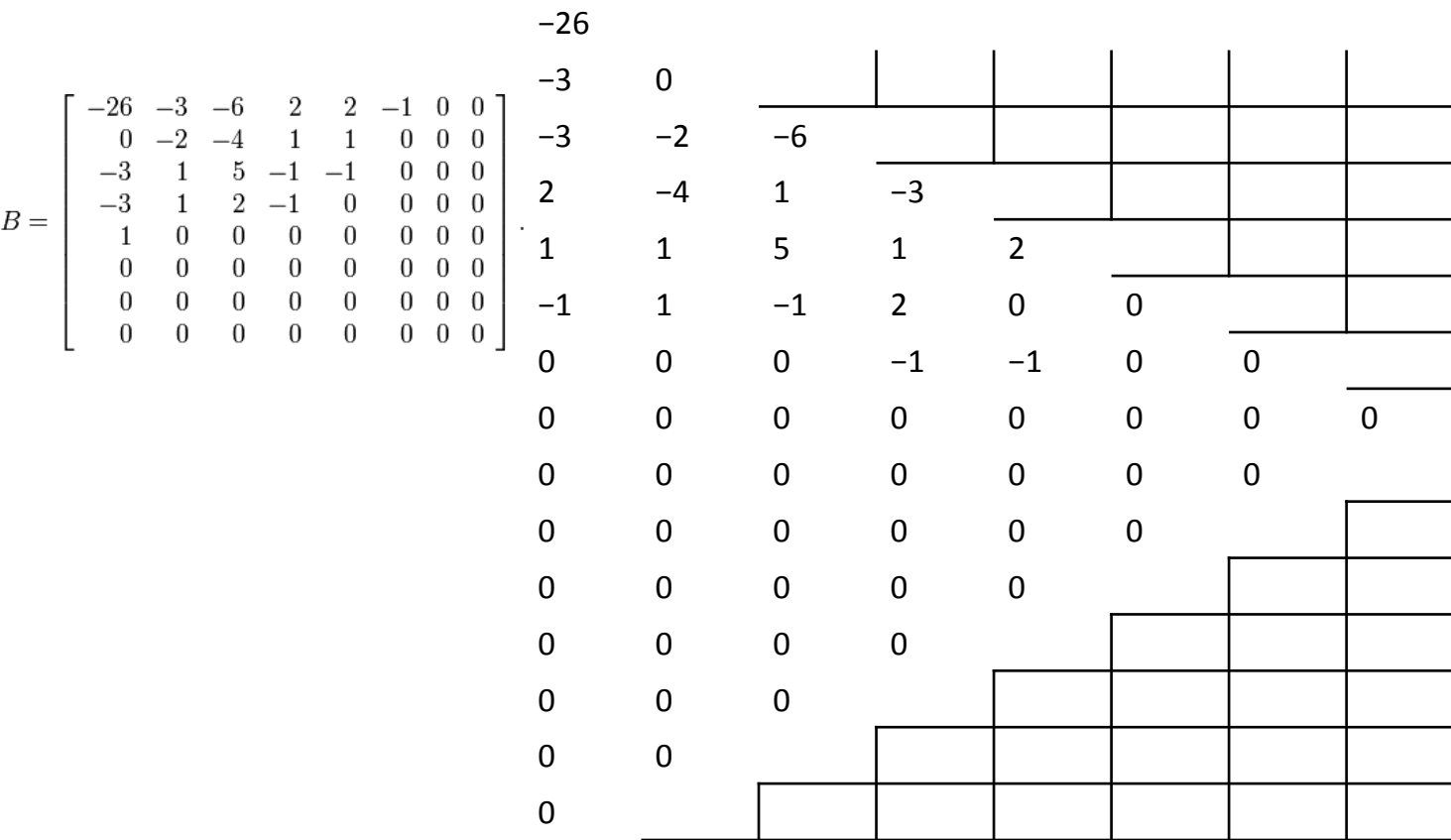
To perform decoding, read the encoded bit stream, and traverse the Huffman tree from the root node toward a leaf node, following the path as indicated by the bit stream. For example, if you read in 11, you would travel to the leaf node R. When you reach a leaf node, append its associated symbol to the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

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**.



Run length encoding

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

- run-length (number of zeros before this coefficient),
- size (no. of bits to store the Huffman code for the coefficient),
- actual Huffman code of the coefficient

4 bits each

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 put 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							



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

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 (this is sometimes called as de-quantization).
- 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 8×8 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.**

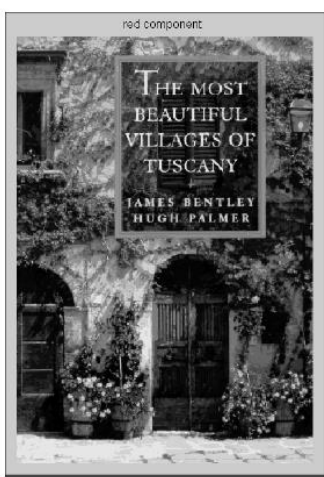
JPEG on Color Images

Color image spaces

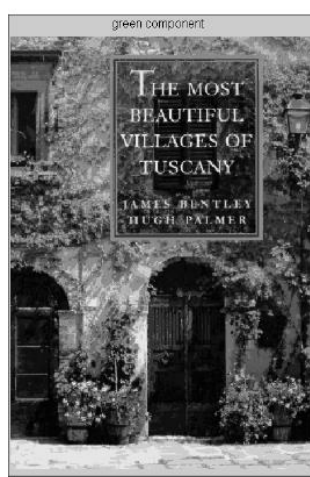
- Color values in a color image are most commonly defined as an (R,G,B) triple, i.e. in the RGB color space.
- There are several other color spaces in which the colors can be represented, eg: HSV, YCbCr, Lab, Luv, CMY(K) and so on.

JPEG for color images

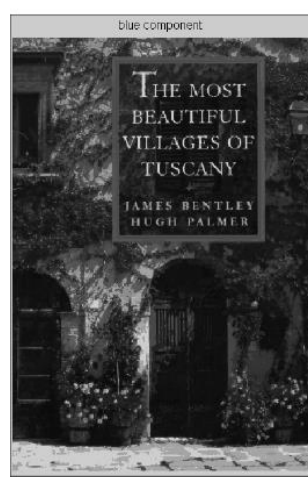
- The RGB values are converted to YCbCr using:
$$Y = 16 + (65.481R + 128.553G + 24.966B)$$
$$C_B = 128 + (-37.79R - 74.203G + 112B)$$
$$C_R = 128 + (112R - 93.786G - 18.214B)$$
- Encode the Y, Cb, and Cr channels separately, using the “grayscale” JPEG algorithm on each channel. The Cb and Cr channels (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.



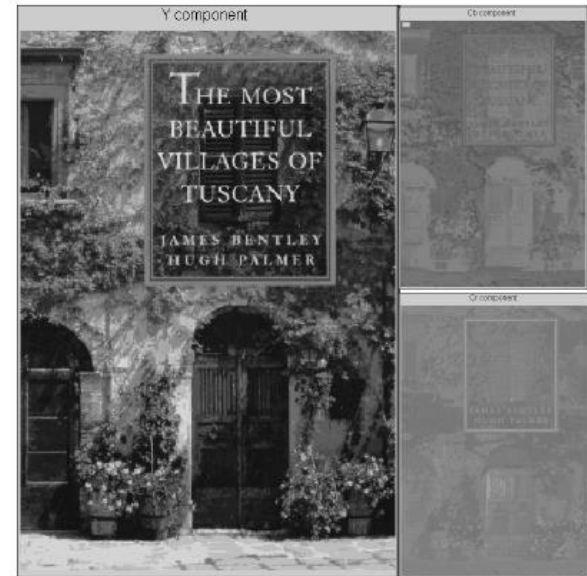
R



G



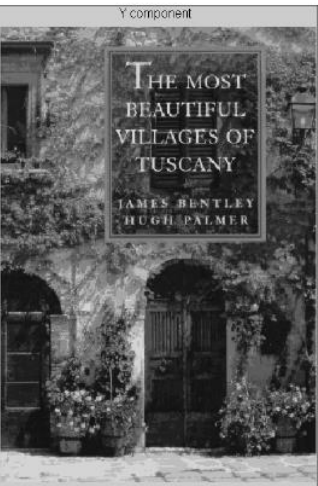
B



Cb

Cr

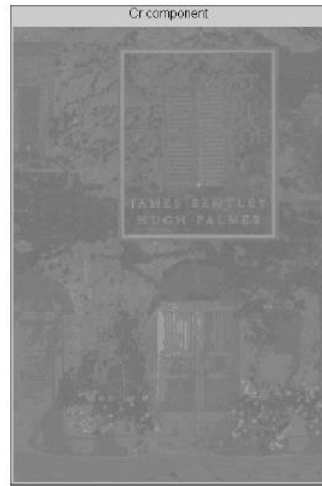
Y



Y



Cb



Cr

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

Down-sampling of chrominance channel in X and Y directions by a factor of 2



4:1:1



4:2:0



4:2:2



4:4:4



Cb channel



PCA on RGB values

- Suppose you take N color images and extract RGB values of each pixel (3×1 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)**.

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 denoted by Y, 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

- 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 $M \times N \times 3$, it stores Y with full resolution (i.e. as an $M \times N$ image), and Cb and Cr with 25% resolution, i.e. as $M/2 \times N/2$ 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

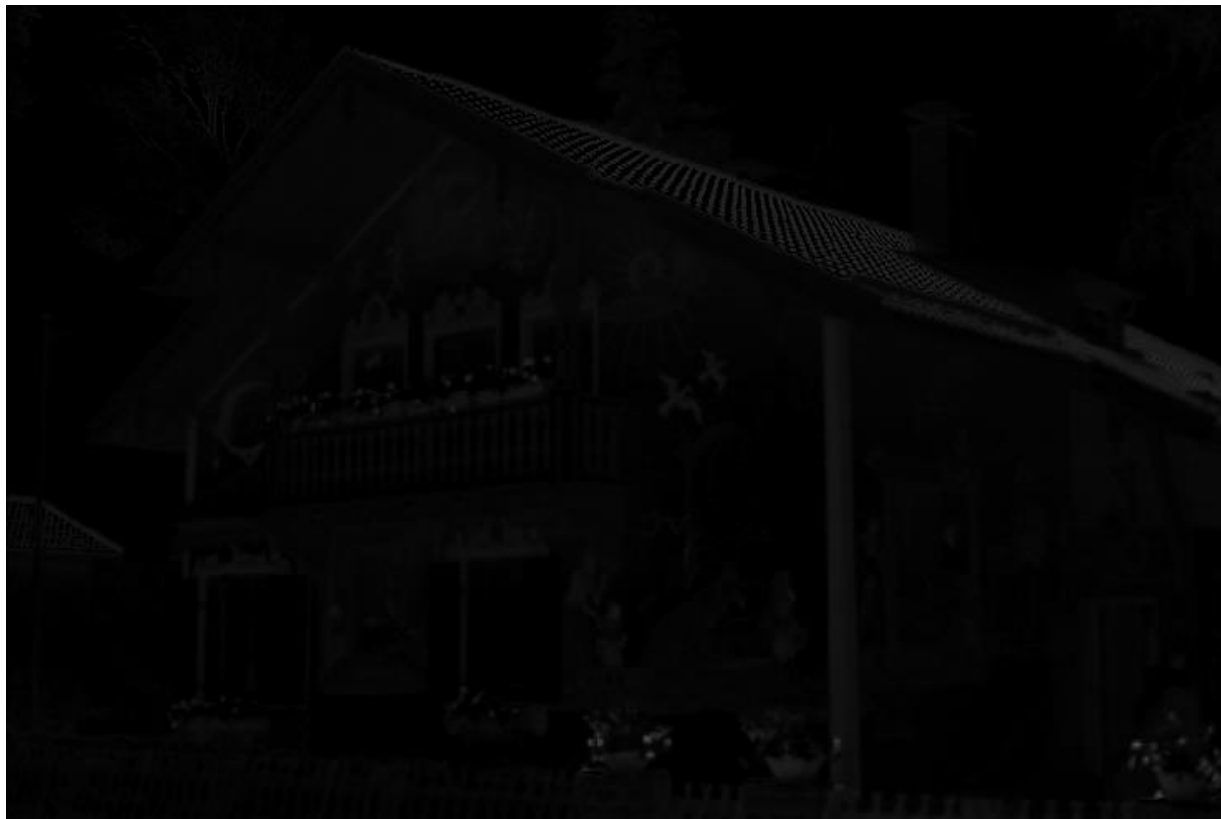
Y channel



Cb channel



Cr channel



RGB, YCbCr and correlation

- The R, G and B values of an image are typically highly correlated, as evidence by high values of the normalized correlation coefficient:

$$\rho = \frac{|(\mathbf{f} - \mu_f) \bullet (\mathbf{g} - \mu_g)|}{\|(\mathbf{f} - \mu_f)\| \|(\mathbf{g} - \mu_g)\|}$$

- It is suboptimal to encode R,G,B separately.
- The corresponding Y, Cb, Cr values are far less correlated, and can be independently encoded at lower costs of storage.

Modes of JPEG Compression

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

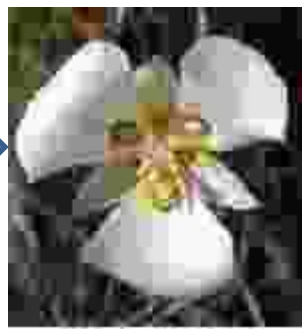
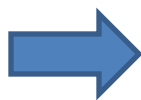


Commonly seen in
web applications (e.g.:
Facebook)

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

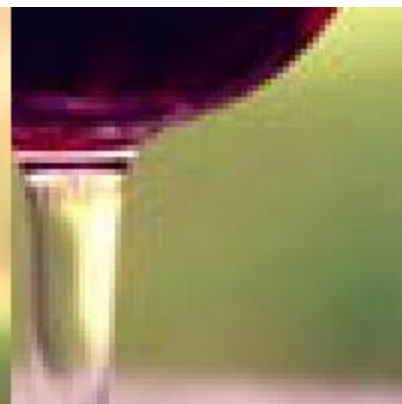
JPEG artifacts

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



Text Caption

<http://www.sitepoint.com/sharper-gif-jpeg-png-images/>



3X

JPEG artifacts are more noticeable when zoomed in.

1X (Left)

A word about JPEG 2000

- JPEG2000 (extension jp2) is the latest series of standards from the JPEG committee
 - Uses wavelet transform
 - Better compression than JPG
 - Superior lossless compression
 - Supports large images and images with many components
 - Region-of-interest coding



JPEG: DCT



JPEG 2000: Wavelets

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

References

- Image compression chapter of the book by Gonzalez
- Section 2.9 and 5.6 of the book by Anil Jain
- Wikipedia article on JPEG
- Image compression slides by Prof. Prem Kalra (IIT Delhi):
<http://www.cse.iitd.ernet.in/~pkalra/siv864/pdf/session-11-4.pdf>