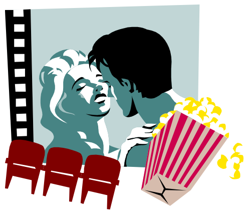
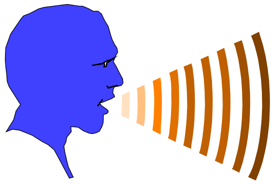



# Image and Video Processing

## Image Processing Based on Orthonormal Transforms

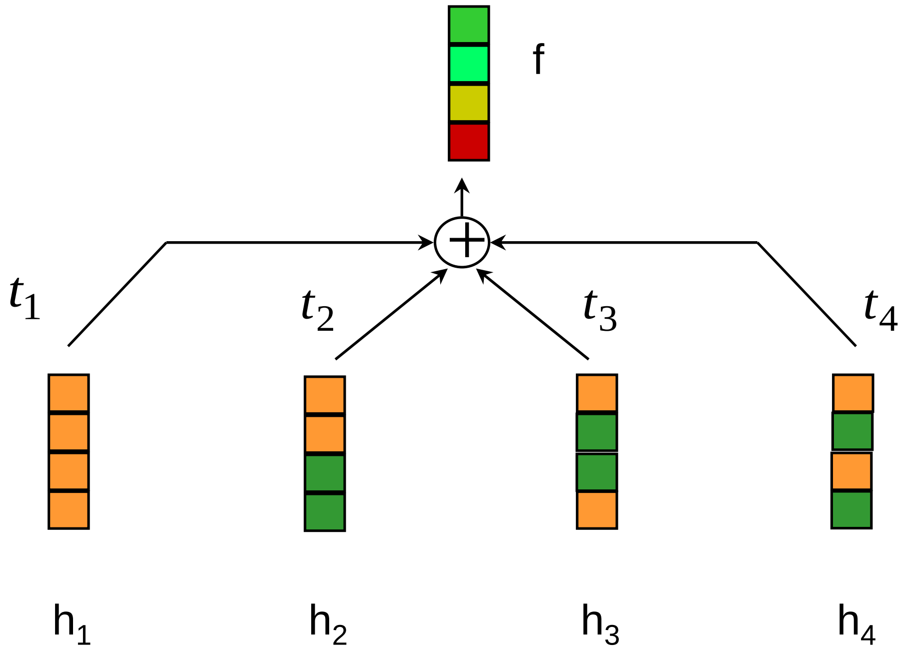
Yao Wang
Tandon School of Engineering, New York University

## Outline

- Unitary transform: from 1D to 2D
- Optimal transform: KLT=PCA
- Redundant transform: dictionary

# Transform Representation (1D)

- Represent a vector f as the weighted combination of some basis vectors $h_n$ . Weights $t_n$ are called transform coefficients
- An N-dimensional vector needs N non-linearly dependent bases



# One Dimensional Linear Transform

- Let $C^N$ represent the N dimensional complex space.
- Let $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{N-1}$ represent N linearly independent vectors in $C^N$.
- Any vector $\mathbf{f} \in C^N$ can be represented as a linear combination of $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{N-1}$ :

$$\mathbf{f} = \sum_{k=0}^{N-1} t(k)\mathbf{h}_k = \mathbf{B}\mathbf{t},$$

$$\textit{where} \quad \mathbf{B} = [\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{N-1}], \mathbf{t} = \begin{bmatrix} t(0) \\ t(1) \\ \vdots \\ t(N-1) \end{bmatrix}.$$

$$\Rightarrow \quad \mathbf{t} = \mathbf{B}^{-1}\mathbf{f} = \mathbf{A}\mathbf{f}$$

f and t form a transform pair

# Inner Product

- Definition of inner product $$<\mathbf{f}_1,\mathbf{f}_2>=\mathbf{f}_1^{H}\mathbf{f}_2=\sum_{n=0}^{N-1} f_1^{*}(n)\, f_2(n)$$
- Orthogonal $$<\mathbf{f}_1,\mathbf{f}_2>=0$$
- 2-Norm of a vector $$\|\mathbf{f}\|^2=<\mathbf{f},\mathbf{f}>=\mathbf{f}^{H}\mathbf{f}=\sum_{n=0}^{N-1}|f(n)|^2$$
- Normalized vector: unit norm $$\|\mathbf{f}\|^2=1$$
- Orthonomal = orthogonal + normalized

## Orthonormal Basis Vectors (OBV)

- $\{\mathbf{h}_k, k=0,\dots N-1\}$ are OBV if

$$<\mathbf{h}_k, \mathbf{h}_l> = \delta_{k,l} = \begin{cases} 1 & k = l \\ 0 & k \neq l \end{cases}$$

- With OBV

$$<\mathbf{h}_l, \mathbf{f}> = <\mathbf{h}_l, \sum_{k=0}^{N-1} t(k)\mathbf{h}_k> = \sum_{k=0}^{N-1} t(k) <\mathbf{h}_l, \mathbf{h}_k> = t(l) = \mathbf{h}_l^H \mathbf{f}$$

$$\mathbf{t} = \begin{bmatrix} \mathbf{h}_0^H \\ \mathbf{h}_1^H \\ \vdots \\ \mathbf{h}_{N-1}^H \end{bmatrix} \mathbf{f} = \mathbf{B}^H \mathbf{f} = \mathbf{A}\mathbf{f}$$

$$\mathbf{B}^{-1} = \mathbf{B}^H, \ or\ \mathbf{B}^H\mathbf{B} = \mathbf{B}\mathbf{B}^H = \mathbf{I}.$$

**B** is unitary

# Definition of Unitary Transform

- Basis vectors are orthonormal
- Forward transform

$$t(k) = <\mathbf{h}_k, \mathbf{f}> = \sum_{n=0}^{N-1} h_k(n)^* f(n),$$

$$\mathbf{t} = \begin{bmatrix} \mathbf{h}_0^H \\ \mathbf{h}_1^H \\ \vdots \\ \mathbf{h}_{N-1}^H \end{bmatrix} \mathbf{f} = \mathbf{B}^H \mathbf{f} = \mathbf{A}\mathbf{f}$$

- Inverse transform

$$f(n) = \sum_{k=0}^{N-1} t(k) h_k(n),$$

$$\mathbf{f} = \sum_{k=0}^{N-1} t(k)\mathbf{h}_k = [\mathbf{h}_0 \quad \mathbf{h}_1 \quad \cdots \quad \mathbf{h}_{N-1}]\mathbf{t} = \mathbf{B}\mathbf{t} = \mathbf{A}^H \mathbf{t}$$

# Example: 4-pt Hadamard Transform

$$\mathbf{h}_0 = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}, \mathbf{h}_1 = \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ -1/2 \end{bmatrix}, \mathbf{h}_2 = \begin{bmatrix} 1/2 \\ -1/2 \\ -1/2 \\ 1/2 \end{bmatrix}, \mathbf{h}_3 = \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix},$$

$$\mathbf{f} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} t_0 = 5 \\ t_1 = -2 \\ t_2 = 0 \\ t_3 = -1 \end{cases}$$

# 1D DFT as a Unitary Transform

$$F(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f(n) e^{-j2\pi \frac{kn}{N}}, \quad k = 0, 1, \ldots, N-1;$$

$$f(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} F(k) e^{j2\pi \frac{kn}{N}}, \quad n = 0, 1, \ldots, N-1.$$

$$h_k(n) = \frac{1}{\sqrt{N}} e^{j2\pi \frac{kn}{N}}, \quad or$$

$$\mathbf{h}_k = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ e^{j2\pi \frac{k}{N}} \\ \vdots \\ e^{j2\pi \frac{(N-1)k}{N}} \end{bmatrix}, k = 0,1,\ldots, N-1.$$

# Example: 1D DFT, N=2

$N=2$ case: there are only two basis vectors:

$$\mathbf{h}_k = \frac{1}{\sqrt{2}}\begin{bmatrix} \exp(j2\pi \frac{k}{2}0) \\ \exp(j2\pi \frac{k}{2}1) \end{bmatrix}: \quad \mathbf{h}_0 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{h}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

if $\mathbf{f} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, determine $t_0, t_1$

Using $t_k = <\mathbf{h}_k, \mathbf{f}>$, we obtain

$$t_0 = \frac{1}{\sqrt{2}}\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) = \frac{1}{\sqrt{2}}(1*1+1*2) = \frac{3}{\sqrt{2}}, t_1 = \frac{1}{\sqrt{2}}\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) = \frac{1}{\sqrt{2}}(1*1-1*2) = \frac{-1}{\sqrt{2}}$$

$$\text{Verify}: t_0\mathbf{h}_0 + t_1\mathbf{h}_1 = \frac{3}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \mathbf{f}$$

# Another Example: 1D DFT, N=4

$$N = 4 \text{ case: using } \mathbf{h}_k = \frac{1}{2}\begin{bmatrix} \exp(j2\pi \frac{k}{4} 0) \\ \exp(j2\pi \frac{k}{4} 1) \\ \exp(j2\pi \frac{k}{4} 2) \\ \exp(j2\pi \frac{k}{4} 3) \end{bmatrix} \text{ yields: } \mathbf{h}_0 = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}; \mathbf{h}_1 = \frac{1}{2}\begin{bmatrix} 1 \\ j \\ -1 \\ -j \end{bmatrix}; \mathbf{h}_2 = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}; \mathbf{h}_3 = \frac{1}{2}\begin{bmatrix} 1 \\ -j \\ -1 \\ j \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 3 \end{bmatrix} \Rightarrow \begin{array}{l} t_0 = \frac{1}{2}(2+4+5+3) = 7; \quad t_1 = \frac{1}{2}(2-4j-5+3j) = -\frac{1}{2}(3+j); \\ t_2 = \frac{1}{2}(2-4+5-3) = 0; \quad t_3 = \frac{1}{2}(2+4j-5-3j) = -\frac{1}{2}(3-j). \end{array}$$

$$\text{Verify: } t_0\mathbf{h}_0 + t_1\mathbf{h}_1 + t_2\mathbf{h}_2 + t_3\mathbf{h}_3 = \frac{1}{4}\begin{bmatrix} 14-(3+j)-(3-j) \\ 14-(3+j)j+(3-j)j \\ 14+(3+j)+(3-j) \\ 14+(3+j)j-(3-j)j \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 8 \\ 16 \\ 20 \\ 12 \end{bmatrix} = \mathbf{f}$$

# 1D Discrete Cosine Transform (DCT)

Basis Vectors:

$$h_k(n) = \alpha(k)\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

$$\text{where } \alpha(k) = \begin{cases} \sqrt{1/N} & k = 0 \\ \sqrt{2/N} & k = 1,\ldots,N-1 \end{cases}$$

$$\text{Forward Transform: } T(k) = \sum_{n=0}^{N-1} f(n)h_k(n)$$

$$\text{Inverse Transforms: } f(n) = \sum_{u=0}^{N-1} T(k)h_k(n)$$

Vector Representation $N = 4$ case:

$$\mathbf{h}_k = \alpha(k)\begin{bmatrix} \cos\left(\frac{1}{8}k\pi\right) \\ \cos\left(\frac{3}{8}k\pi\right) \\ \cos\left(\frac{5}{8}k\pi\right) \\ \cos\left(\frac{7}{8}k\pi\right) \end{bmatrix} \text{ yields: } \mathbf{h}_0 = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}; \mathbf{h}_1 = \sqrt{\frac{1}{2}}\begin{bmatrix} \cos\left(\frac{1}{8}\pi\right) \\ \cos\left(\frac{3}{8}\pi\right) \\ \cos\left(\frac{5}{8}\pi\right) \\ \cos\left(\frac{7}{8}\pi\right) \end{bmatrix} = \begin{bmatrix} 0.6533 \\ 0.2706 \\ -0.2706 \\ -0.6533 \end{bmatrix}; \mathbf{h}_2 = \sqrt{\frac{1}{2}}\begin{bmatrix} \cos\left(\frac{2}{8}\pi\right) \\ \cos\left(\frac{6}{8}\pi\right) \\ \cos\left(\frac{10}{8}\pi\right) \\ \cos\left(\frac{14}{8}\pi\right) \end{bmatrix} = \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix}; \mathbf{h}_3 = \sqrt{\frac{1}{2}}\begin{bmatrix} \cos\left(\frac{3}{8}\pi\right) \\ \cos\left(\frac{9}{8}\pi\right) \\ \cos\left(\frac{15}{8}\pi\right) \\ \cos\left(\frac{21}{8}\pi\right) \end{bmatrix} = \begin{bmatrix} 0.2706 \\ -0.6533 \\ 0.6533 \\ -0.2706 \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 3 \end{bmatrix} \Rightarrow \begin{aligned} &t_0 = \frac{1}{2}(2+4+5+3) = 7; \quad t_1 = (2-3)*0.6533 + (4-5)*0.2706 = -0.9239; \\ &t_2 = \frac{1}{2}(2-4-5+3) = -2; \quad t_3 = (2-3)*0.2706 + (5-4)*0.6533 = 0.3827. \end{aligned}$$

Verify: $t_0\mathbf{h}_0 + t_1\mathbf{h}_1 + t_2\mathbf{h}_2 + t_3\mathbf{h}_3 = \mathbf{f}$

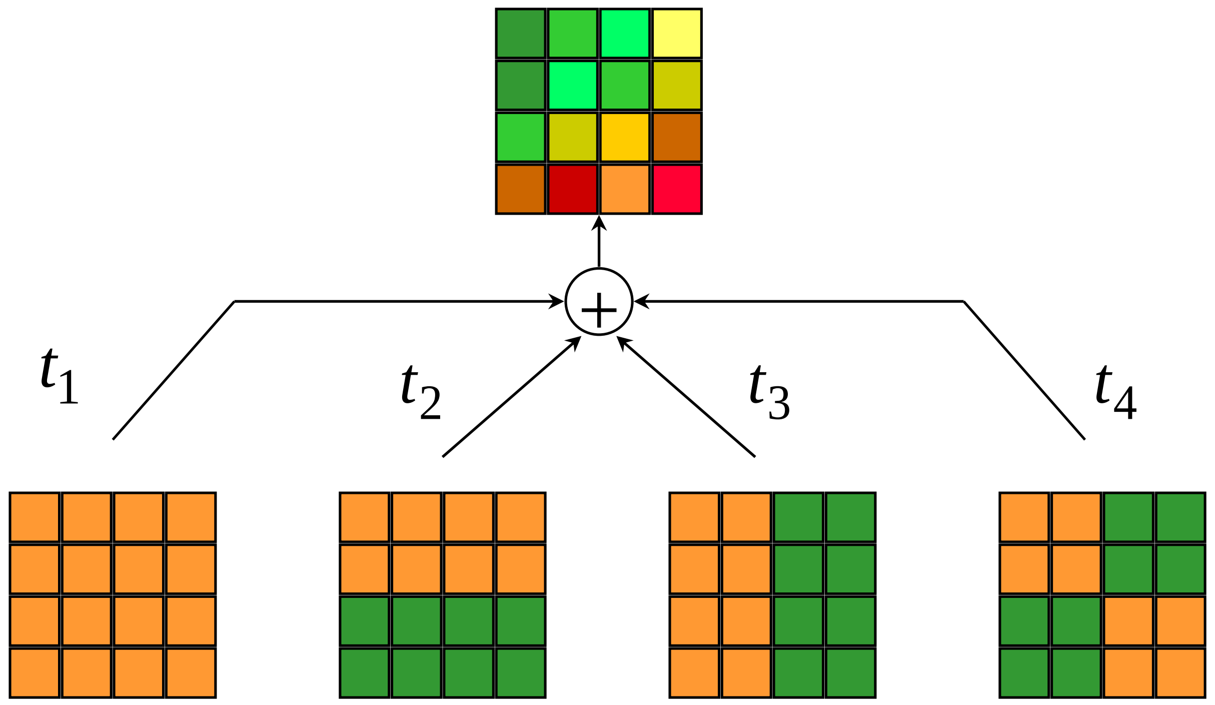
## Two Dimensional Transform

- Decompose a MxN 2D matrix F=[F(m,n)] into a linear combination of some basic images, $H_{k,l}=[H_{k,l}(m,n)]$, so that:

$$\mathbf{F} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} T(k,l)\mathbf{H}_{k,l},$$

$$F(m,n) = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} T(k,l)H_{k,l}(m,n)$$

# Transform Representation of an Image Block



Inverse transform: Represent a block of image samples as the superposition of some basic block patterns (basis images)
Forward transform: Determine the coefficients associated with each basis image

## 2D Transform Can be Treated as 1D Transform

- Arrange each image block into a vector
- Arrange each basis image as a vector using the same order
- But this does not take advantage of special properties of Separable Transform

## Separable Unitary Transform

- Let $\mathbf{h}_k$, k=0, 1, …, M-1 represent orthonormal basis vectors in $C^M$,
- Let $\mathbf{g}_l$, l=0, 1, …, N-1 represent orthonormal basis vectors in $C^N$,
- Let $\mathbf{H}_{k,l}=\mathbf{h}_k\mathbf{g}_l^T$, or $H_{k,l}(m,n)=h_k(m)g_l(n)$.
- Then $\mathbf{H}_{k,l}$ will form an orthonormal basis set in $C^{MxN}$.

# Example of Separable Unitary Transform

- Example 1

$$\mathbf{h}_0 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{h}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}.$$

$$\mathbf{H}_{00} = \mathbf{h}_0 \mathbf{h}_0^T = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix} \qquad \mathbf{H}_{01} = \mathbf{h}_0 \mathbf{h}_1^T = \begin{bmatrix} 1/2 & -1/2 \\ 1/2 & -1/2 \end{bmatrix}$$

$$\mathbf{H}_{10} = \mathbf{h}_1 \mathbf{h}_0^T = \begin{bmatrix} 1/2 & 1/2 \\ -1/2 & -1/2 \end{bmatrix} \qquad \mathbf{H}_{11} = \mathbf{h}_1 \mathbf{h}_1^T = \begin{bmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{bmatrix}$$

# Example: 4x4 DFT

$$\text{Recall the 1D DFT basis are:}\quad \mathbf{h}_0=\frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix};\mathbf{h}_1=\frac{1}{2}\begin{bmatrix}1\\j\\-1\\-j\end{bmatrix};\mathbf{h}_2=\frac{1}{2}\begin{bmatrix}1\\-1\\1\\-1\end{bmatrix};\mathbf{h}_3=\frac{1}{2}\begin{bmatrix}1\\-j\\-1\\j\end{bmatrix}$$

using $\mathbf{H}_{k,l}=\mathbf{h}_k(\mathbf{h}_l)^T$ yields:

$$\mathbf{H}_{0,0}=\frac{1}{4}\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix},\quad \mathbf{H}_{0,1}=\frac{1}{4}\begin{bmatrix}1&j&-1&-j\\1&j&-1&-j\\1&j&-1&-j\\1&j&-1&-j\end{bmatrix},\quad \mathbf{H}_{0,2}=\frac{1}{4}\begin{bmatrix}1&-1&1&-1\\1&-1&1&-1\\1&-1&1&-1\\1&-1&1&-1\end{bmatrix},\quad \mathbf{H}_{0,3}=\frac{1}{4}\begin{bmatrix}1&-j&-1&j\\1&-j&-1&j\\1&-j&-1&j\\1&-j&-1&j\end{bmatrix}$$

$$\mathbf{H}_{1,0}=\frac{1}{4}\begin{bmatrix}1&1&1&1\\j&j&j&j\\-1&-1&-1&-1\\-j&-j&-j&j\end{bmatrix},\mathbf{H}_{1,1}=\frac{1}{4}\begin{bmatrix}1&j&-1&-j\\j&-1&-j&1\\-1&-j&1&j\\-j&1&j&-1\end{bmatrix},\mathbf{H}_{1,2}=\frac{1}{4}\begin{bmatrix}1&-1&1&-1\\j&-j&j&-j\\-1&1&-1&1\\-j&j&-j&j\end{bmatrix},\mathbf{H}_{1,3}=\frac{1}{4}\begin{bmatrix}1&-j&-1&j\\j&1&-j&-1\\-1&j&1&-j\\-j&-1&j&1\end{bmatrix}$$

$$\mathrm{H}_{2,0}=\frac{1}{4}\begin{bmatrix}1&1&1&1\\-1&-1&-1&-1\\1&1&1&1\\-1&-1&-1&-1\end{bmatrix},\quad \mathbf{H}_{2,1}=\frac{1}{4}\begin{bmatrix}1&j&-1&-j\\-1&-j&1&j\\1&j&-1&-j\\-1&-j&1&j\end{bmatrix},\quad \mathbf{H}_{2,2}=\frac{1}{4}\begin{bmatrix}1&-1&1&-1\\-1&1&-1&1\\1&-1&1&-1\\-1&1&-1&1\end{bmatrix},\quad \mathbf{H}_{2,3}=\frac{1}{4}\begin{bmatrix}1&-j&-1&j\\-1&j&1&-j\\1&-j&-1&j\\-1&j&1&-j\end{bmatrix}$$

$$\mathbf{H}_{3,0}=\frac{1}{4}\begin{bmatrix}1&1&1&1\\-j&-j&-j&-j\\-1&-1&-1&-1\\j&j&j&j\end{bmatrix},\mathbf{H}_{3,1}=\frac{1}{4}\begin{bmatrix}1&j&-1&-j\\-j&1&j&-1\\-1&-j&1&j\\j&-1&-j&1\end{bmatrix},\mathbf{H}_{3,2}=\frac{1}{4}\begin{bmatrix}1&-1&1&-1\\-j&j&-j&j\\-1&1&-1&1\\j&-j&j&-j\end{bmatrix},\mathbf{H}_{3,3}=\frac{1}{4}\begin{bmatrix}1&-j&-1&j\\-j&-1&j&1\\-1&j&1&-j\\j&1&-j&-1\end{bmatrix}$$

## Example: 4x4 DFT

For $\mathbf{F} = \begin{bmatrix} 1 & 2 & 2 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & -1 \end{bmatrix}$, compute $T_{k,l}$

Using $T_{k,l} =< \mathbf{H}_{k,l}, \mathbf{F} >$ yields, e.g,

$$T_{0,0} =< \mathbf{H}_{0,0}, \mathbf{F} >= \frac{1}{4}\left( \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 2 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & -1 \end{bmatrix} \right) = \frac{1}{4}(1+2+2+0+0+1+3+1+0+1+2+1+1+2+2-1) = \frac{18}{4}$$

$$T_{2,3} =< \mathbf{H}_{2,3}, \mathbf{F} >= \frac{1}{4}\left( \begin{bmatrix} 1 & -j & -1 & j \\ -1 & j & 1 & -j \\ 1 & -j & -1 & j \\ -1 & j & 1 & -j \end{bmatrix}^{*}, \begin{bmatrix} 1 & 2 & 2 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & -1 \end{bmatrix} \right) = \frac{1}{4}(1+2j-2-j+3+j+j-2-j-1-2j+2-j) = \frac{1}{4}(1-j)$$

# Example: 8x8 DCT

$$H_{k,l}(m,n) = \alpha(k)\alpha(l)\cos\left[\frac{(2m+1)k\pi}{2N}\right]\cos\left[\frac{(2n+1)l\pi}{2N}\right]$$

$$where\,\alpha(k) = \begin{cases} \sqrt{1/N} & k=0 \\ \sqrt{2/N} & k=1,\ldots,N-1 \end{cases}$$

Low-Low

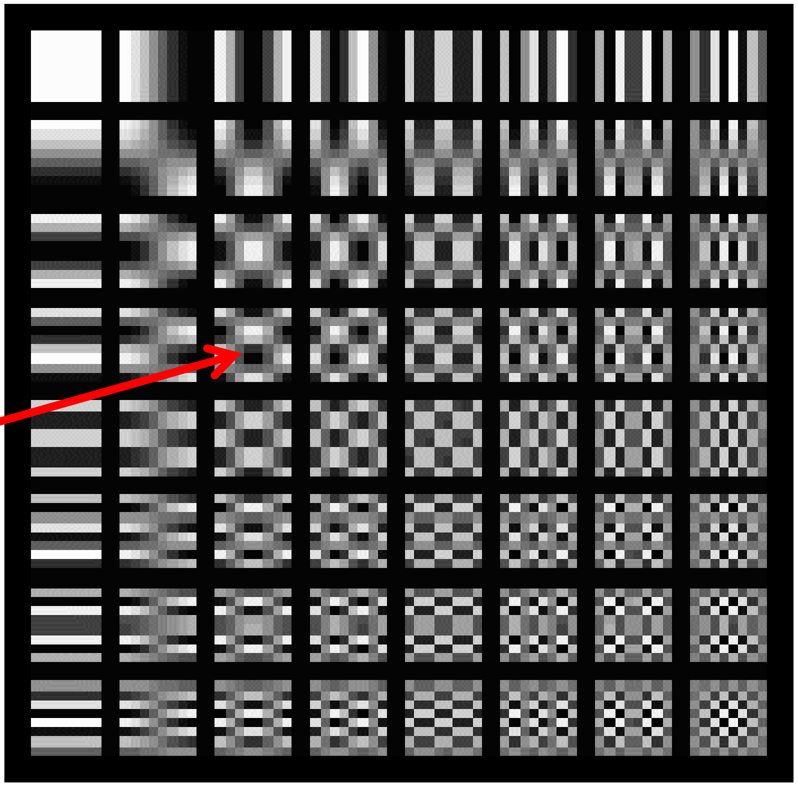
High-Low

Basis Images:

D=dctmtx(8);
Basis43=D(:,4)*D(3,:)';

Low-High

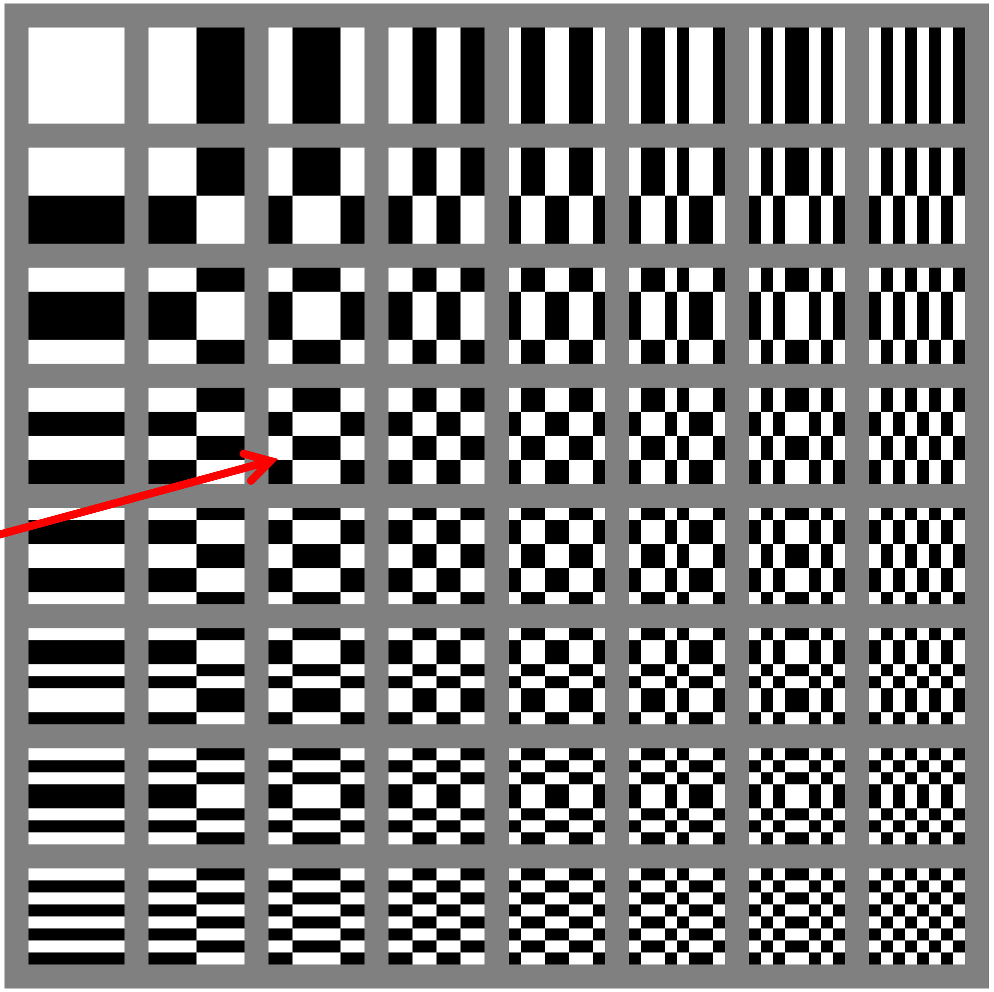
High-High



# Hadamard Transform: Basis images

Example:

```
D=hadamard(8);
reindex=[1,8,4,5,2,7,3,6];
D(reindex,:)=D;
Basis43=D(:,4)*D(:,3)';
```



From Amy Reibman

# Property of Separable Transform

- When the transform is separable, we can perform the 2D transform separately.
  - First, do 1D transform for each row using basis vectors $g_l$,
  - Second, do 1D transform for each column of the intermediate image using basis vectors $h_k$.
  - Proof:

$$T(k,l)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}H_{k,l}^{*}(m,n)F(m,n)=\sum_{m=0}^{M-1}h_{k}^{*}(m)\sum_{n=0}^{N-1}g_{l}^{*}(n)F(m,n)=\sum_{m=0}^{M-1}h_{k}^{*}(m)U(m,l)$$

# DCT on a Real Image Block

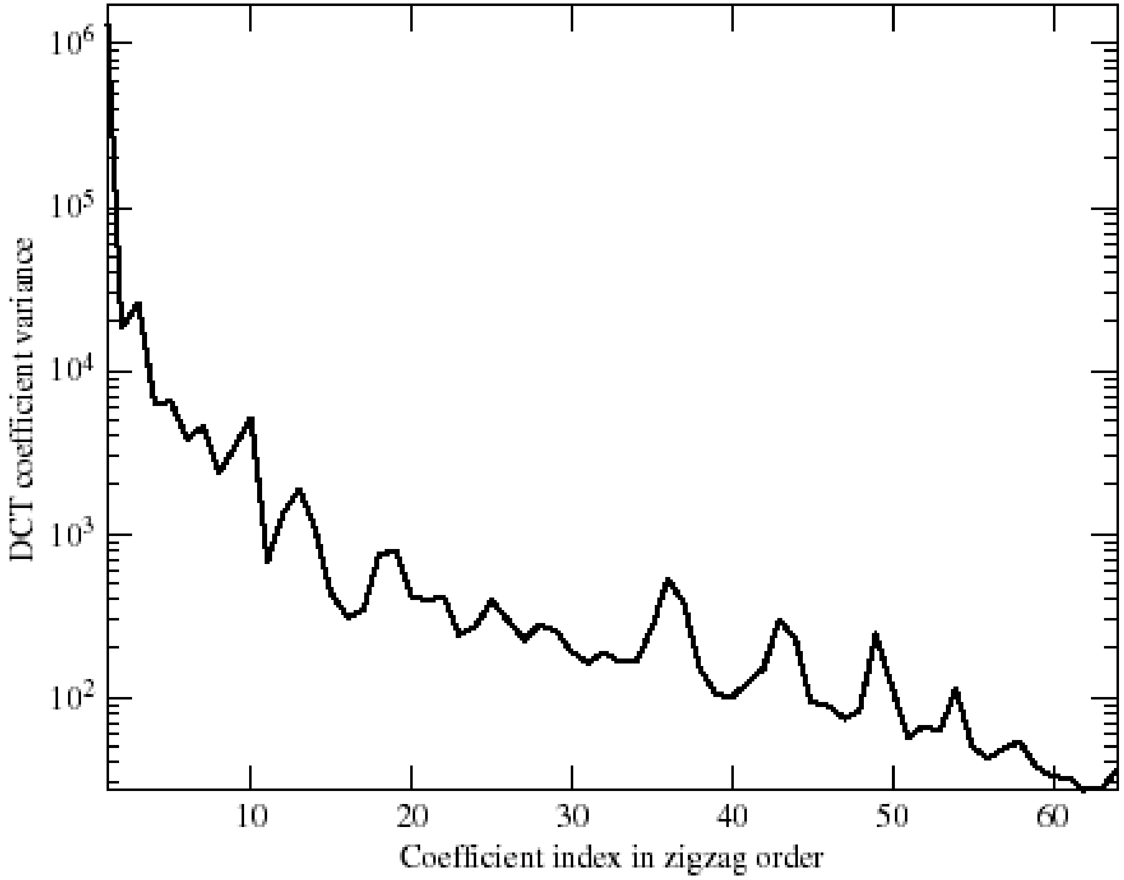
```
>>imblock = lena256(128:135,128:135)-128
imblock=
 54    68    71    73    75    73    71    45
   47    52    48    14    20    24    20    -8
   20   -10    -5   -13   -14   -21   -20   -21
  -13   -18   -18   -16   -23   -19   -27   -28
  -24   -22   -22   -26   -24   -33   -30   -23
  -29   -13     3   -24   -10   -42   -41     5
  -16    26    26   -21    12   -31   -40    23
   17    30    50    -5     4    12    10     5
```

```
>>dctblock =dct2(imblock)
dctblock=
 31.0000   51.7034    1.1673  -24.5837  -12.0000  -25.7508   11.9640    23.2873
  113.5766    6.9743  -13.9045   43.2054   -6.0959   35.5931  -13.3692  -13.0005
  195.5804   10.1395   -8.6657   -2.9380  -28.9833   -7.9396    0.8750     9.5585
   35.8733  -24.3038  -15.5776  -20.7924   11.6485  -19.1072   -8.5366  0.5125
   40.7500  -20.5573  -13.6629   17.0615  -14.2500   22.3828   -4.8940  -11.3606
    7.1918  -13.5722   -7.5971  -11.9452   18.2597  -16.2618   -1.4197    -3.5087
   -1.4562  -13.3225   -0.8750    1.3248   10.3817   16.0762    4.4157    1.1041
   -6.7720   -2.8384    4.1187    1.1118   10.5527   -2.7348   -3.2327    1.5799
```

Note that low-low coefficients are much larger than high-high coefficients

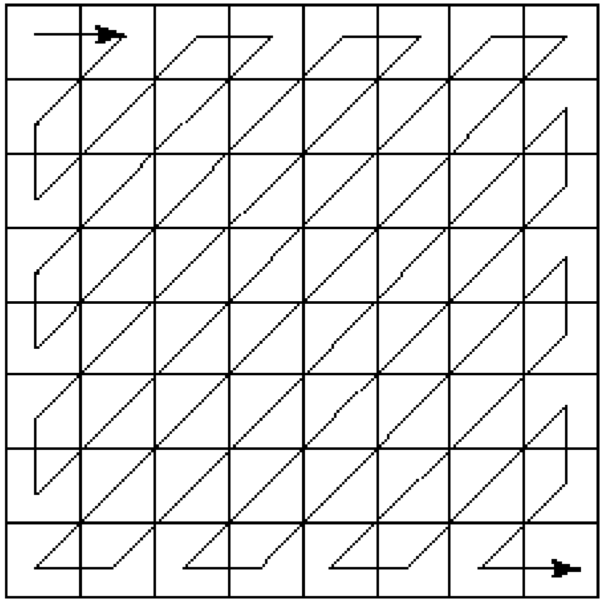
We subtract 128 from original image block shift the mean to zero

# Energy Distribution of DCT Coefficients in Typical Image Blocks

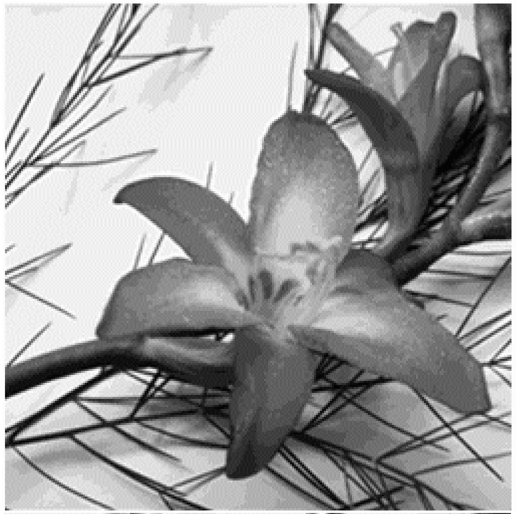


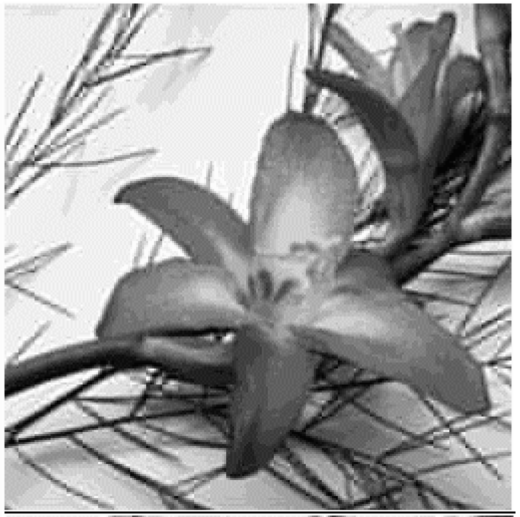
Variance of each coefficient is determined by the average of the square of this coefficient in all blocks of an image

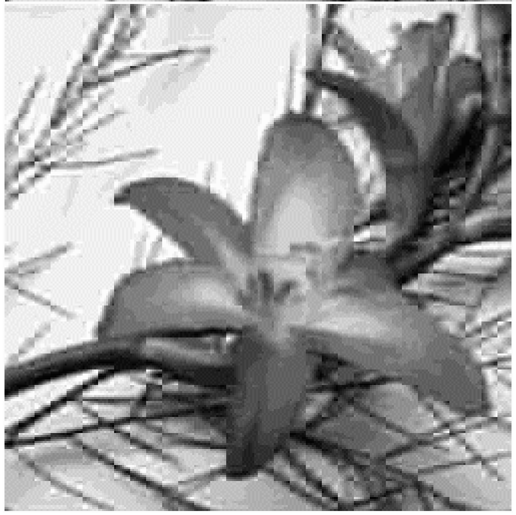
Zig-zag ordering

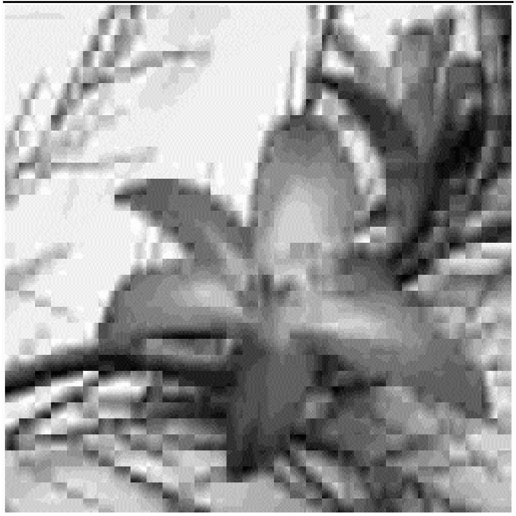


## Images Approximated by Different Number of DCT Coefficients per Block


Original


With 16/64 Coefficients

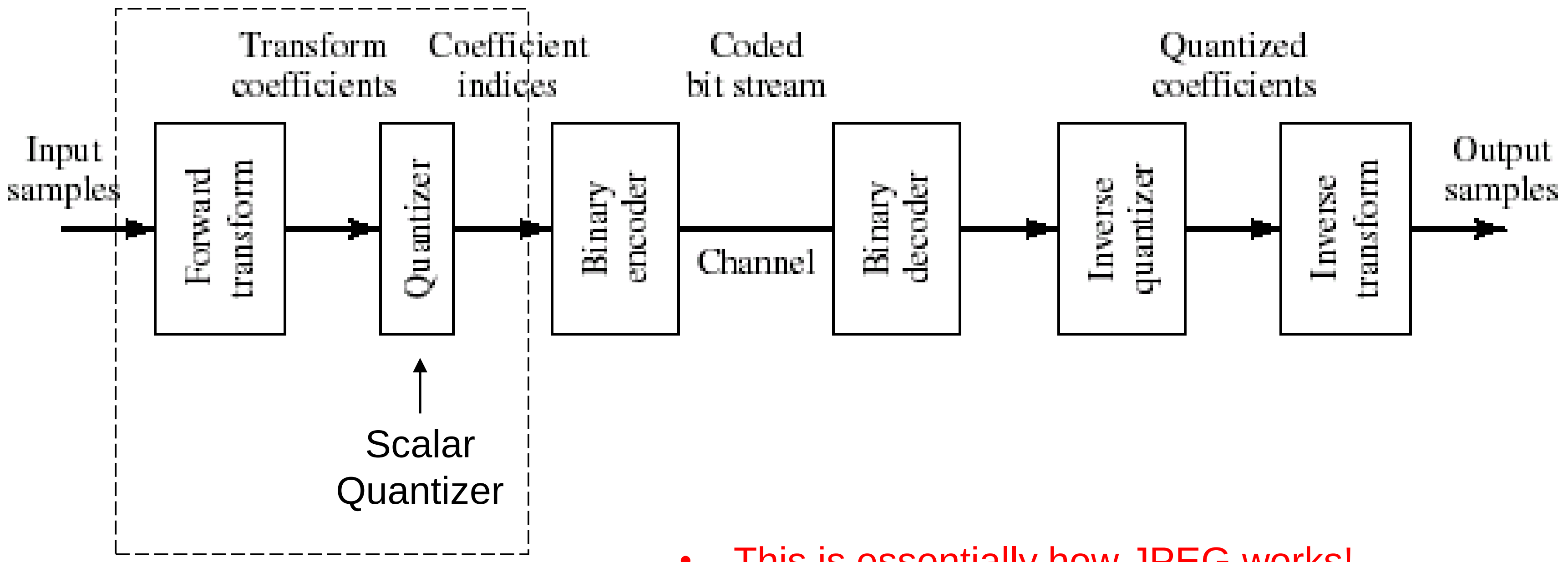

With 8/64 Coefficients


With 4/64 Coefficients

## Why do we want to use transform?

- **Compression:** With a well design set of basis vectors, only a few coefficients of typical image blocks are large, and coefficients are uncorrelated. Each image block can be represented by its quantized transform coefficients!
- **Feature dimension reduction:** Using a few large coefficients rather than original samples.
- **Denoising:** Noise typically contribute to small coefficients at all frequencies, but real data typically have significant coefficients only at low frequencies. We can suppress noise by setting small high frequency coefficients to zero.
- Just as DFT, we could attenuate different DCT coefficients to achieve different filtering effect (but the convolution property does not hold for DCT)

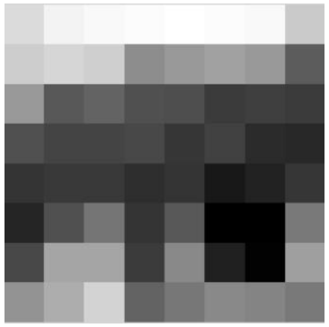
# Block Diagram of Transform Coding

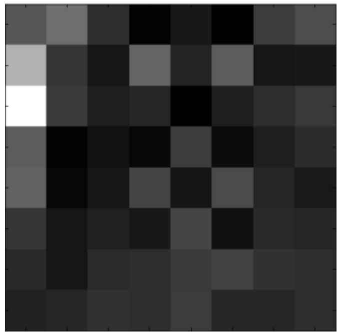


- This is essentially how JPEG works!
- JPEG uses 8x8 DCT, and Runlength+Huffman coding for binary encoding.

# JPEG Example: DCT on a Real Image Block

```
>>imblock = lena256(128:135,128:135)-128
imblock=
 54    68    71    73    75    73    71    45
   47    52    48    14    20    24    20    -8
   20   -10    -5   -13   -14   -21   -20   -21
  -13   -18   -18   -16   -23   -19   -27   -28
  -24   -22   -22   -26   -24   -33   -30   -23
  -29   -13     3   -24   -10   -42   -41     5
  -16    26    26   -21    12   -31   -40    23
   17    30    50    -5     4    12    10     5
```



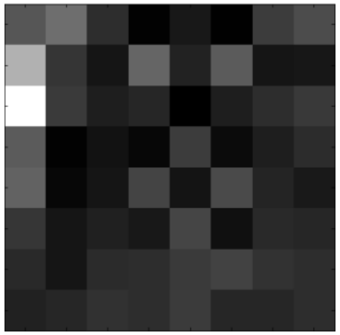


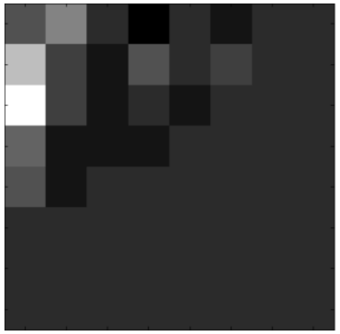
```
>>dctblock =dct2(imblock)
dctblock=
 31.0000   51.7034    1.1673  -24.5837  -12.0000  -25.7508   11.9640    23.2873
  113.5766    6.9743  -13.9045   43.2054   -6.0959   35.5931  -13.3692  -13.0005
  195.5804   10.1395   -8.6657   -2.9380  -28.9833   -7.9396    0.8750     9.5585
   35.8733  -24.3038  -15.5776  -20.7924   11.6485  -19.1072   -8.5366  0.5125
   40.7500  -20.5573  -13.6629   17.0615  -14.2500   22.3828   -4.8940  -11.3606
    7.1918  -13.5722   -7.5971  -11.9452   18.2597  -16.2618   -1.4197    -3.5087
   -1.4562  -13.3225   -0.8750    1.3248   10.3817   16.0762    4.4157    1.1041
   -6.7720   -2.8384    4.1187    1.1118   10.5527   -2.7348   -3.2327    1.5799
```

In JPEG, “imblock-128” is done before DCT to shift the mean to zero

# JPEG Example: Quantized Indices

```
>>dctblock =dct2(imblock)
dctblock=
 31.0000   51.7034    1.1673  -24.5837  -12.0000  -25.7508   11.9640   23.2873
 113.5766    6.9743  -13.9045   43.2054   -6.0959   35.5931  -13.3692  -13.0005
 195.5804   10.1395   -8.6657   -2.9380  -28.9833   -7.9396    0.8750    9.5585
  35.8733  -24.3038  -15.5776  -20.7924   11.6485  -19.1072   -8.5366   0.5125
  40.7500  -20.5573  -13.6629   17.0615  -14.2500   22.3828   -4.8940  -11.3606
   7.1918  -13.5722   -7.5971  -11.9452   18.2597  -16.2618   -1.4197   -3.5087
  -1.4562  -13.3225   -0.8750    1.3248   10.3817   16.0762    4.4157    1.1041
  -6.7720   -2.8384    4.1187    1.1118   10.5527   -2.7348   -3.2327    1.5799
```



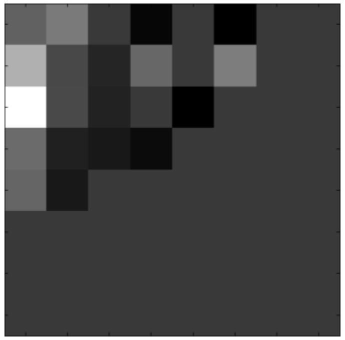


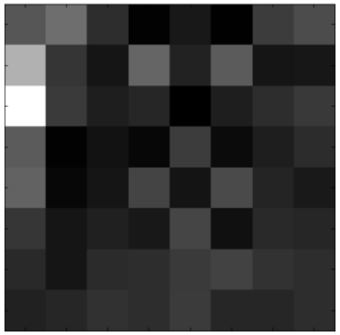
```
>>QP=1;
>>QM=Qmatrix*QP;
>>qdct=floor((dctblock+QM/2)./(QM))
qdct =
    2    5    0   -2    0   -1    0    0
    9    1   -1    2    0    1    0    0
   14    1   -1    0   -1    0    0    0
    3   -1   -1   -1    0    0    0    0
    2   -1    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
```

Only 19 coefficients are retained out of 64

# JPEG Example: Quantized Coefficients

```
%dequantized DCT block
>>  iqdct=qdct.*QM
iqdct=
      32      55       0     -32       0     -40    0   0
     108      12     -14      38       0      58    0   0
     196      13     -16       0     -40       0    0   0
      42     -17     -22     -29       0       0    0   0
      36     -22       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
```



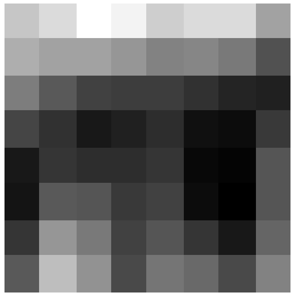


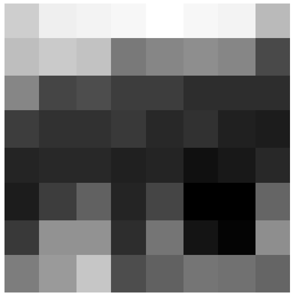
```
Original DCT block
dctblock=
 31.0000   51.7034    1.1673  -24.5837  -12.0000  -25.7508   11.9640    23.2873
  113.5766    6.9743  -13.9045   43.2054   -6.0959   35.5931  -13.3692  -13.0005
  195.5804   10.1395   -8.6657   -2.9380  -28.9833   -7.9396    0.8750     9.5585
   35.8733  -24.3038  -15.5776  -20.7924   11.6485  -19.1072   -8.5366   0.5125
   40.7500  -20.5573  -13.6629   17.0615  -14.2500   22.3828   -4.8940  -11.3606
    7.1918  -13.5722   -7.5971  -11.9452   18.2597  -16.2618   -1.4197    -3.5087
   -1.4562  -13.3225   -0.8750    1.3248   10.3817   16.0762    4.4157    1.1041
   -6.7720   -2.8384    4.1187    1.1118   10.5527   -2.7348   -3.2327    1.5799
```

# JPEG Example: Reconstructed Image

```
%reconstructed image block
>>  qimblock=round(idct2(iqdct))
qimblock=
    58    68    85    79    61    68    67    38
    45    38    39    33    22    24    19    -2
    21     2   -11   -12   -13   -19   -24   -27
    -8   -19   -31   -26   -20   -35   -37   -15
   -31   -17   -21   -20   -16   -39   -41     0
   -33     3    -1   -14   -11   -37   -44     1
   -16    32    18   -10     1   -16   -30     8
     3    54    30    -6    16    11    -7    23
```

```
Original image block
imblock=
    54    68    71    73    75    73    71    45
    47    52    48    14    20    24    20    -8
    20   -10    -5   -13   -14   -21   -20   -21
   -13   -18   -18   -16   -23   -19   -27   -28
   -24   -22   -22   -26   -24   -33   -30   -23
   -29   -13     3   -24   -10   -42   -41     5
   -16    26    26   -21    12   -31   -40    23
    17    30    50    -5     4    12    10     5
```





## Transform design

- What are desirable properties of a transform for image and video?
  - Decorrelating
    - Good for compression: one can entropy code each coefficient independently without losing efficiency;
    - Good for feature reduction: each feature reveal independent info.
  - High energy compaction – Only a few large coefficients, rest are zero or negligible
  - Easy to compute (few operations)
  - Separable – compute 1-D transform first on rows, then on columns. So we only consider design of 1-D transform bases.
- What size transform should we use?
  - Entire image? Small blocks?
  - 2-D (on an image) or 3-D (incorporating time also)?

## Karhunen Loève Transform (KLT) = Principle Component Analysis (PCA)

- Basis vectors in directions with largest variance = eigenvectors (principle components) of the signal covariance matrix.
- Coefficients are completely uncorrelated ☺
- Best energy compaction ☺
  - Sort coefficients from largest to smallest in expected squared magnitude; then the sum of the energies of the first M coefficients is as large as possible
- No computationally efficient algorithm ☹
- Requires the knowledge of the mean and covariance matrix of the signal vector. ☹

## Properties of Unitary Transforms

**s**=[s$_n$]: signal vector; **t**=[t$_k$]: transform coefficient vector; **u**$_k$: Basis vectors

forward transform: $t_k = \langle \mathbf{u}_k, \mathbf{s} \rangle$ or $\mathbf{t} = [\mathbf{U}]^H \mathbf{s} = [\mathbf{V}]\mathbf{s}$

inverse transform: $\mathbf{s} = \sum_{k \in \mathcal{N}} t_k \mathbf{u}_k = [\mathbf{U}]\mathbf{t} = [\mathbf{V}]^H \mathbf{t}.$

**1.** The mean vectors $\boldsymbol{\eta}_s = E\{\boldsymbol{\mathcal{S}}\}$ and $\boldsymbol{\eta}_t = E\{\boldsymbol{\mathcal{T}}\}$ are related by

$$\boldsymbol{\eta}_t = [\mathbf{V}]\boldsymbol{\eta}_s, \qquad \boldsymbol{\eta}_s = [\mathbf{V}]^H \boldsymbol{\eta}_t. \tag{9.1.11}$$

The covariance matrices $[\mathbf{C}]_s = E\{(\boldsymbol{\mathcal{S}} - \boldsymbol{\eta}_s)(\boldsymbol{\mathcal{S}} - \boldsymbol{\eta}_s)^H\}$ and $[\mathbf{C}]_t = E\{(\boldsymbol{\mathcal{T}} - \boldsymbol{\eta}_t)(\boldsymbol{\mathcal{T}} - \boldsymbol{\eta}_t)^H\}$ are related by

$$[\mathbf{C}]_t = [\mathbf{V}][\mathbf{C}]_s[\mathbf{V}]^H, \qquad [\mathbf{C}]_s = [\mathbf{V}]^H[\mathbf{C}]_t[\mathbf{V}]. \tag{9.1.12}$$

From Wang, et al, Digital video processing and communications, 2002.

# Properties of Unitary Transforms

**2.** The total energy of the transformed vector equals that of the sample vector. This is true both for a given realization and the ensemble average; that is,

$$\sum_{n \in \mathcal{N}} s_n^2 = \sum_{k \in \mathcal{N}} t_k^2, \tag{9.1.13}$$

$$\sum_{n \in \mathcal{N}} \sigma_{s,n}^2 = \sum_{k \in \mathcal{N}} \sigma_{t,k}^2. \tag{9.1.14}$$

where $\sigma_{s,n}^2 = E\{(\mathcal{S}_n - \eta_{s,n})^2\}$ and $\sigma_{t,k}^2 = E\{(\mathcal{T}_k - \eta_{t,k})^2\}$ are the variances of $\mathcal{S}_n$ and $\mathcal{T}_k$, respectively. (This property is equivalent to Parseval's theorem for the Fourier transform.)

From Wang, et al, Digital video processing and communications, 2002.

# Properties of Unitary Transforms

**3.** Suppose that we use only the first $K < N$ coefficients to approximate $\mathbf{s}$, with the approximated vector being $\hat{\mathbf{s}}_K = \sum_{k=1}^{K} t_k \mathbf{u}_k$; then the approximation error signal is $\mathbf{e}_K = \mathbf{s} - \hat{\mathbf{s}}_K = \sum_{k=K+1}^{N} t_k \mathbf{u}_k$. The approximation error energy for a given $\mathbf{s}$ is

$$\|\mathbf{e}_K\|^2 = \sum_{n \in \mathcal{N}} e_n^2 = \sum_{k=K+1}^{N} t_k^2. \qquad (9.1.15)$$

The variance of $\boldsymbol{\mathcal{E}}_K = \boldsymbol{\mathcal{S}} - \hat{\boldsymbol{\mathcal{S}}}_K$ over the ensemble of $\boldsymbol{\mathcal{S}}$ is

$$E\{\|\boldsymbol{\mathcal{E}}_K\|^2\} = \sum_{n \in \mathcal{N}} \sigma_{e,n}^2 = \sum_{k=K+1}^{N} \sigma_{t,k}^2. \qquad (9.1.16)$$

Because the sum of all coefficient squares or variances is a constant (Equation (9.1.13) or (9.1.14)), the approximation error for a particular signal vector is minimized if one chooses $K$ coefficients that have the largest values to approximate the original signal. Similarly, the mean approximation error is minimized by choosing $K$ coefficients with the largest variances.

From Wang, et al, Digital video processing and communications, 2002.

# Karhunen Loève Transform (KLT)

- KLT: Using eigenvectors of signal covariance matrix $[C]_s$ as transform bases, ordered based on the eigenvalues

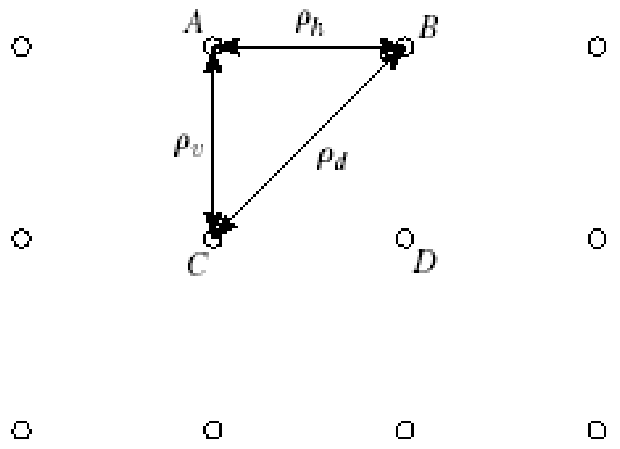
$$[\mathbf{C}]_s \phi_k = \lambda_k \phi_k, \quad \text{with} \quad \langle \phi_k, \phi_l \rangle = \delta_{k,l}.$$

$$\sigma_k^2 = \lambda_k. \qquad [\mathbf{C}]_t = \begin{bmatrix} \lambda_1 & & \\ & \cdots & \\ & & \lambda_N \end{bmatrix}$$

  - Resulting transform coefficients are uncorrelated
  - The coefficient variances are the eigenvalues
  - K-term approximation error = The expected square error of representing a vector with first K coefficients = sum of last N-K eigen-values
  - It can be shown that among all possible unitary transforms, KLT yields the least K-term approximation error, for all K=1,2,…,N.

## Example

- Consider 2x2 image blocks with inter-sample correlation as shown below.



$$\rho_h = \rho_d = \rho, \rho_d = \rho^2.$$

## Example Continued (Convert 2x2 into 4x1)

- Correlation matrix

$$[\mathrm{C}]_s = E\left\{\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}[A \quad B \quad C \quad D]\right\} = \begin{bmatrix} C_{AA} & C_{AB} & C_{AC} & C_{AD} \\ C_{BA} & C_{BB} & C_{BC} & C_{BD} \\ C_{CA} & C_{CB} & C_{CC} & C_{CD} \\ C_{DA} & C_{DB} & C_{DC} & C_{DD} \end{bmatrix}$$

$$= \sigma_s^2 \begin{bmatrix} 1 & \rho_h & \rho_v & \rho_d \\ \rho_h & 1 & \rho_d & \rho_v \\ \rho_v & \rho_d & 1 & \rho_h \\ \rho_d & \rho_v & \rho_h & 1 \end{bmatrix}.$$

- DCT basis images

$$\frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \frac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}, \quad \frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad \frac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

- Equivalent 1D transform matrix

$$[\mathrm{U}] = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

# Example

- Determine the KLT for the 2x2 image block in the previous example

$$[\mathbf{C}]_s = \sigma_s^2 \begin{bmatrix} 1 & \rho_h & \rho_v & \rho_d \\ \rho_h & 1 & \rho_d & \rho_v \\ \rho_v & \rho_d & 1 & \rho_h \\ \rho_d & \rho_v & \rho_h & 1 \end{bmatrix}$$

Determine the eigenvalues by solving: $\det([\mathbf{C}]_s - \lambda[\mathbf{I}]) = 0$

$$\lambda_k = \{(1+\rho)^2, (1-\rho^2), (1-\rho^2), (1-\rho)^2\}\sigma_s^2.$$

(same as the coefficient variances with DCT)

Determine the eigenvectors by solving $([\mathbf{C}]_s - \lambda[\mathbf{I}])\phi_k = 0$

Resulting transform is close to the DCT !
DCT is a good transform for images, with energy compaction close to KLT!
DCT is a fixed transform (does not depend on the signal statistics) and can be computed fast.

## What if I don’t know the covariance matrix?

- Using samples to form the covariance matrix

Given samples $f_k, k = 1,2,\ldots,K$

Mean $\mu_f = \frac{1}{K}\sum_{k=1}^{K} f_k$

Covariance matrix $C_f = \frac{1}{K}\sum_{k=1}^{K} (f_k - \mu_f)(f_k - \mu_f)^T$

Let $\boldsymbol{F}$ be the sample matrix, consisting all the mean removed samples in its columns,

$$\boldsymbol{F} = [f_k - \mu; k = 1,2,\ldots,K]$$

Covariance matrix

$$\boldsymbol{C}_f = \frac{1}{K}\boldsymbol{F}\,\boldsymbol{F}^T$$

## Feature Dimension Reduction by PCA

- Given a set of samples in a dataset, each represented by a raw signal vector of dimension *N*
- Want to find a reduced feature representation of dimension *K*
- Principle component analysis (PCA) = KLT using the sample covariance matrix
  - The eigenvector corresponding to the largest eigen value corresponds to the direction with largest variance.
- Features = first *K* transform coefficients
- Feature dimension reduction is an important problem in machine learning.
- PCA is a linear feature reduction method.
- There are other more powerful non-linear transformations.

# Signal Independent Transform Bases

- Suboptimal transforms – many available!
  - Discrete Fourier Transform (DFT): complex values;
    - convolution in space = multiplication in coefficients;
    - not best in terms of energy compaction.
  - Discrete Cosine transform (DCT): real values only
    - nearly as good as KLT for common image signals
  - Hadamard and Haar: basis functions contain only +1,0,-1
    - Very fast computation
    - Has blocky artifacts with k-term approximation

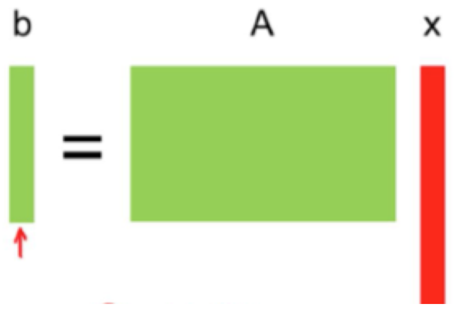
## Pop Quiz

- How do you determine the optimal transform (KLT)?
- What are its properties?
- What are the difficulty of using KLT in practice?
- Why do we use DCT for images?
- How does transform coding work?

# From Transform to Dictionary (Not Required)

- Transform: Using N orthonormal basis vectors to represent a N-vector (Non-redundant dictionary)
  - Have unique solution to Ax=b -> $x=A^T b$
  - Not a good solution if the measured signal has noise!
- Dictionary: Using M (M > N generally) atoms $a_m$, m=1,2,…,M, to represent a N-vector b
  - $b= x_1 a_1 + x_2 a_2 + \ldots + x_M a_M$

$$b_{Nx1} = [a_1 \quad a_2 \quad \cdots \quad a_M] \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_M \end{bmatrix} = A_{NxM} x_{Mx1}$$



- When M>N, determining coefficients is an under-determined problem and cannot be accomplished by a simple matrix inversion!
- Sparse dictionary learning: Given a set of data samples (each a N-vector), determine dictionary atoms that leads to the sparsest representation (with coefficients having minimal number of non-zeros)

## Reading Assignment

- Reading assignment:
  - [Wang2002] Sec. 9.1 (Transform coding)
  - [Gonzalez and Woods 2008] “Digital Image Processing,” Chap 7 (Wavelet transforms), Chap 8 (Image compression)

# Written Homework

1. Answer the quiz questions
2. For the 2x2 image **S** given below, compute its 2D DCT, reconstruct it by retaining different number of coefficients to evaluate the effect of different basis images.
   a) Determine the four DCT basis images.
   b) Show that these basis images are orthonormal to each other.
   c) Determine the 2D-DCT coefficients for **S,** *Tk,l, k=0,1;l=0,1.*
   d) Show that the reconstructed image from the original DFT coefficients equal to the original image.
   e) Modify the DCT coefficients using the given window masks (**W**1 to **W**5) and reconstruct the image using the modified DCT coefficients. (for a given mask, "1" indicates to retain that coefficient, "0" means to set the corresponding coefficient to zero) What effect do you see with each mask and why?

$$\mathbf{S}=\begin{bmatrix}9 & 1\\ 1 & 9\end{bmatrix}, \mathbf{W}_1=\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}, \mathbf{W}_2=\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}, \mathbf{W}_3=\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}, \mathbf{W}_4=\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}, \mathbf{W}_5=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$