ECE 5578 Multimedia Communication

Lec 07 - Transform & Quantization I

Zhu Li

Dept of CSEE, UMKC

Office: FH560E, Email: lizhu@umkc.edu, Ph: x 2346.

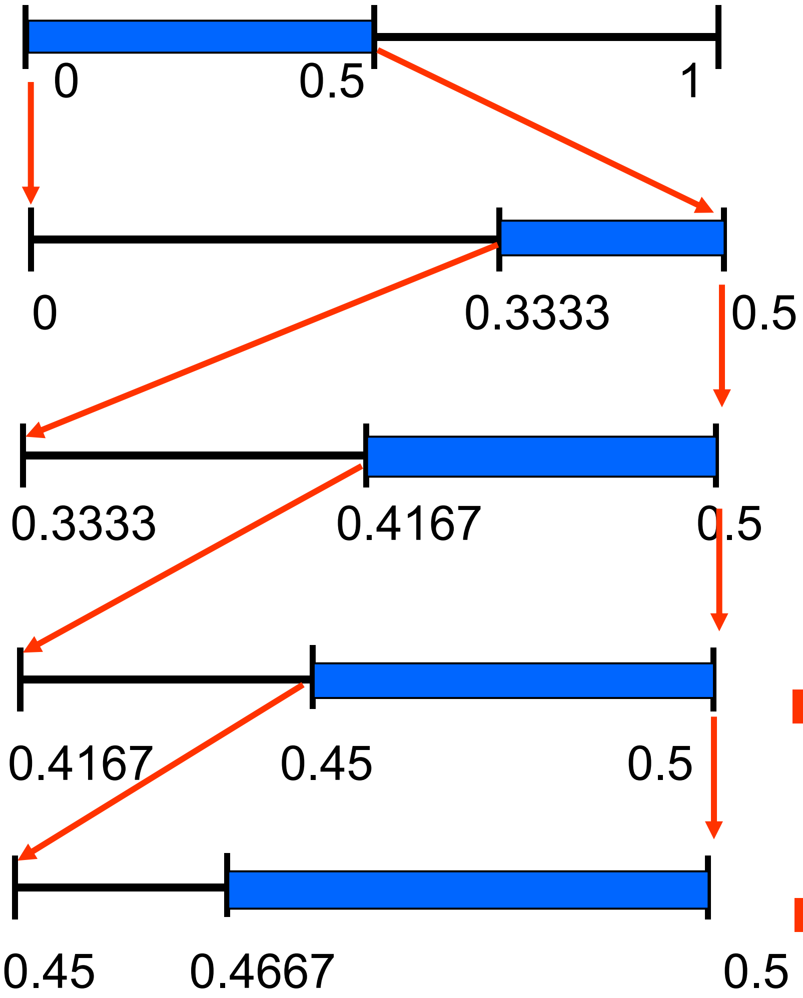
http://l.web.umkc.edu/lizhu

slides created with WPS Office Linux and EqualX LaTex equation editor

Outline

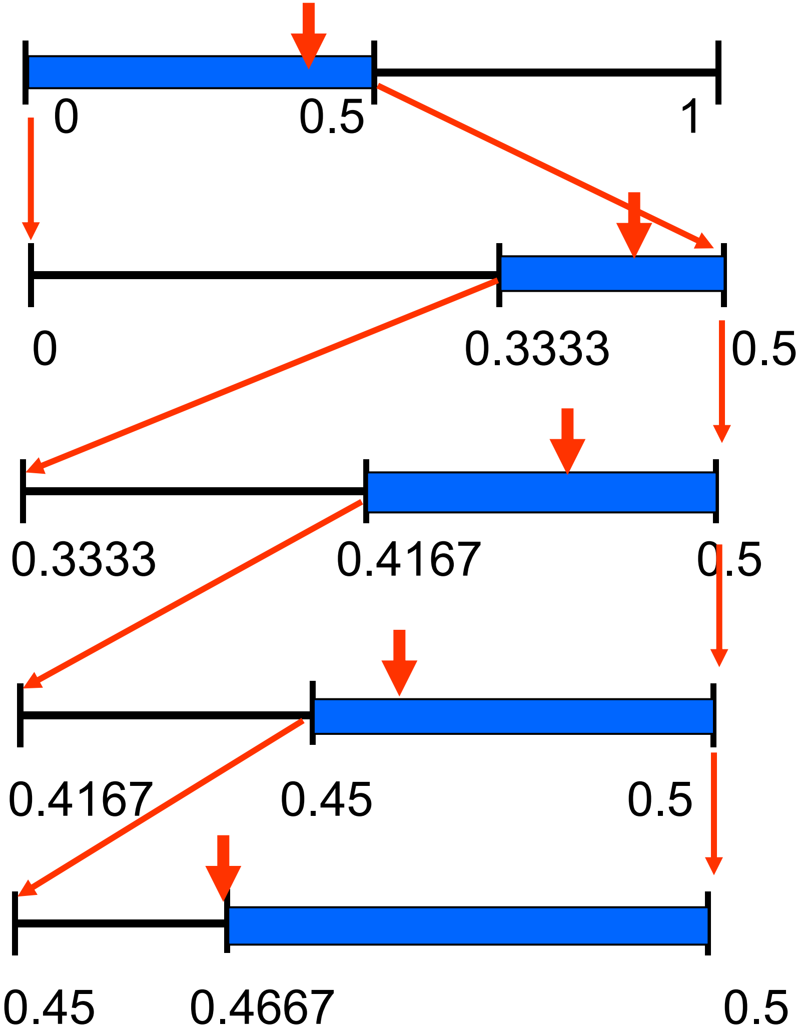
- ❑ Context Model for AC
- ❑ Transform coding
  - Basic Transform Coding Framework
  - KLT/PCA
  - DCT
  - SVD

Context Adaptive AC



- Binary sequence: 01111
- Initial counters for 0's and 1's:
  C(0)=C(1)=1.
  → P(0)=P(1)=0.5
- After encoding 0: C(0)=2, C(1)=1.
  →P(0)=2/3, P(1)=1/3
- After encoding 01: C(0)=2, C(1)=2.
  →P(0)=1/2, P(1)=1/2
- After encoding 011: C(0)=2, C(1)=3.
  →P(0)=2/5, P(1)=3/5
- After encoding 0111: C(0)=2, C(1)=4.
  →P(0)=1/3, P(1)=2/3.
- Encode 0.4667.

Context Adaptive AC Decoding



- Input 0.4667.
- Initial counters for 0's and 1's:

  C(0)=C(1)=1 → P(0)=P(1)=0.5

  Decode 0
- After decoding 0: C(0)=2, C(1)=1. →P(0)=2/3, P(1)=1/3

  Decode 1
- After decoding 01: C(0)=2, C(1)=2. →P(0)=1/2, P(1)=1/2

  Decode 1
- After decoding 011: C(0)=2, C(1)=3. →P(0)=2/5, P(1)=3/5

  Decode 1
- After decoding 0111: C(0)=2, C(1)=4. →P(0)=1/3, P(1)=2/3.

  Decode 1

Modeling Large Context - PAQ

- ❑ Condition reduces entropy, $H(Y|X_1) > H(Y|X_1, X_2, \ldots.)$
- ❑ How to model very large context (x1, x2, ….)?
  - Use a (shallow/deep) neural network to model context
  - Large window of 552 input nodes
  - 7 sets of 3080 hidden nodes

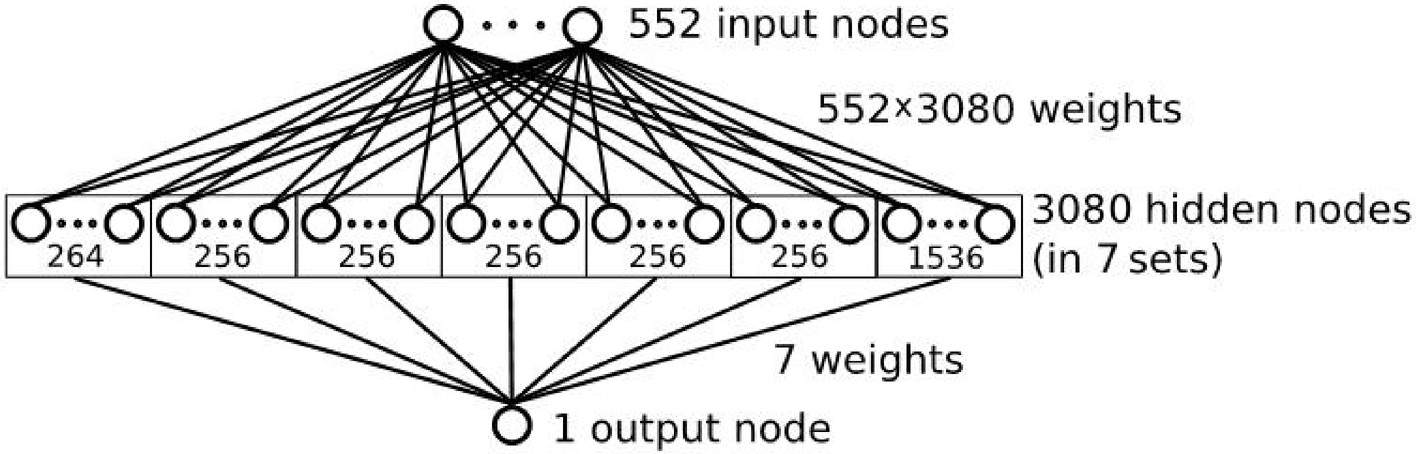
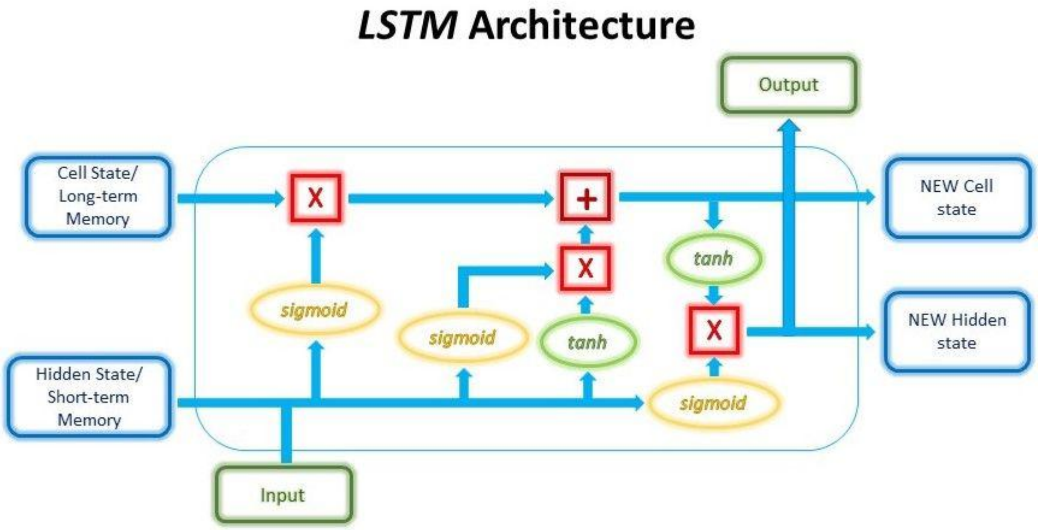


Figure 4: PAQ8 model mixer architecture.

- Potential project: deeper neural network for HEVC CABAC context modeling

LSTM Driven AC Context Model

❑ LSTM



- LSTM tutorial: https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

❑ LSTM + AC:

- use LSTM to read in a sequence of symbols, and then predict the next symbol in sequence, its prob.
- Paras’s LSTM+AC implementation: https://sce.umkc.edu/faculty-sites/lizhu/teaching/2021.spring.video/hw/lstm_ac.zip

Outline

- ❑ Context Model for AC
- ❑ Transform coding
  - Basic Transform Coding Framework
  - KLT/PCA
  - DCT
  - SVD

Transform Coding Framework

❑ Analysis Transform

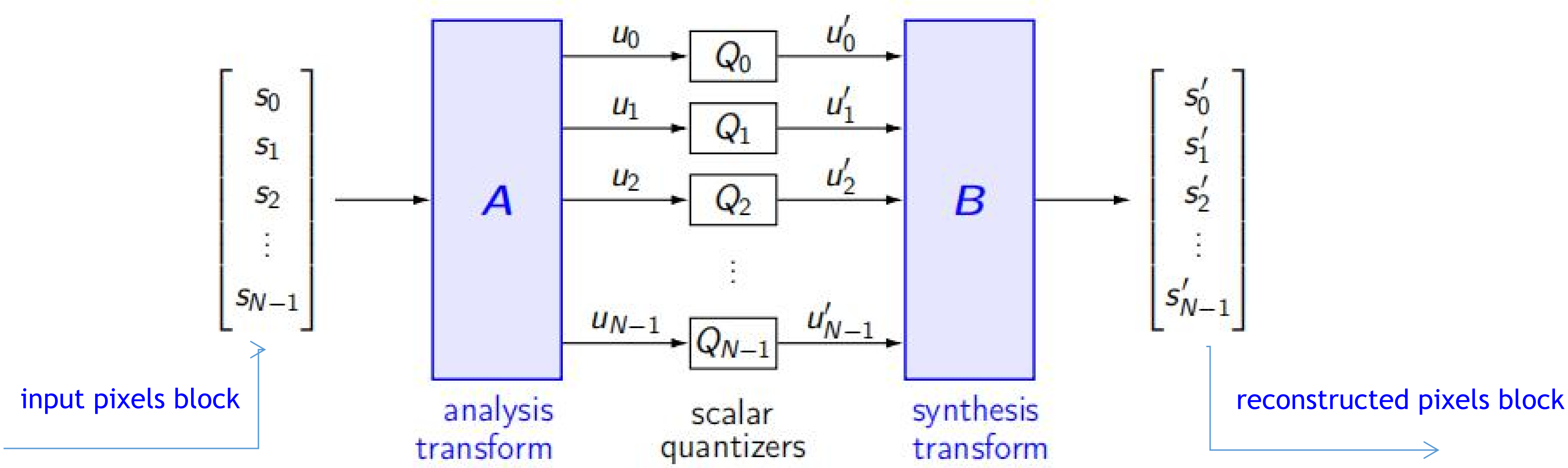
$$U = AS$$

❑ Quantization

$$U' = Q(U)$$

❑ Synthesize Transform

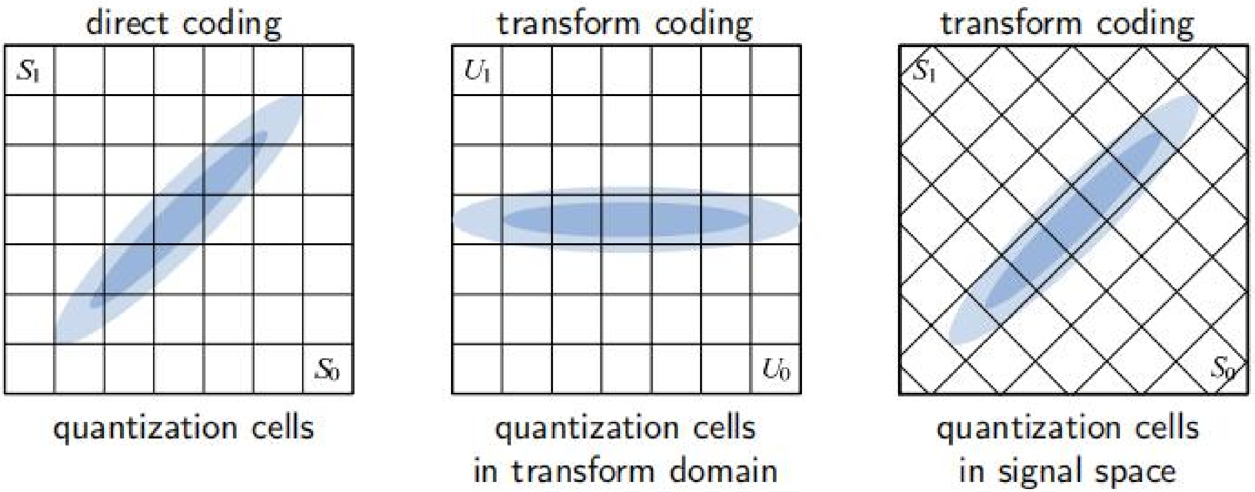
$$S' = BU'$$



Transform Coding

❑ Benefits of Transform Coding

- Remove correlation among pixels
- Compact representation of pixels in transformed coefficients
- Scalar quantization more effective in transform domain
- Complexity: much faster implementation than vector quantization, which is a joint transform-quantization scheme.
- Suitable for perceptual loss modeling, e.g. JPEG quantization table.



Unitary Transform

❑ Definition: Inverse is equal to the conjuate transpose

$$A^{-1} = (A^*)^T$$

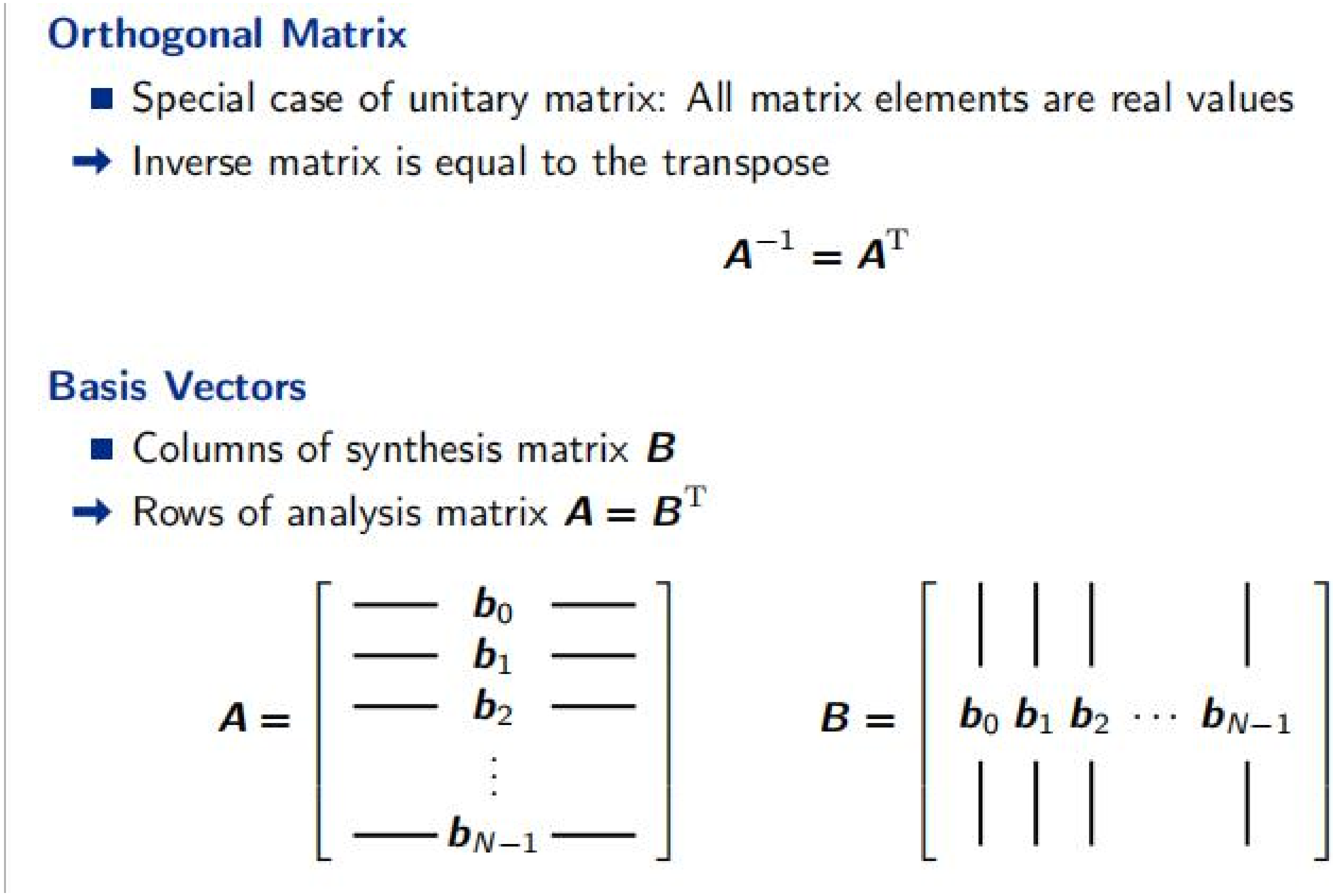
❑Length preserving

$$u = As \qquad \|As\|_2 = \|s\|_2$$

$$\begin{aligned}
\|\boldsymbol{u}\|_2^2 &= \sum_k |u_k|^2 = \sum_k u_k^* \cdot u_k = (\boldsymbol{u}^*)^{\mathrm{T}} \boldsymbol{u} \\
&= \boldsymbol{u}^\dagger \cdot \boldsymbol{u} = (\boldsymbol{A}\boldsymbol{s})^\dagger \cdot (\boldsymbol{A}\boldsymbol{s}) = \boldsymbol{s}^\dagger \cdot \boldsymbol{A}^\dagger \cdot \boldsymbol{A} \cdot \boldsymbol{s} \\
&= \boldsymbol{s}^\dagger \cdot \left(\boldsymbol{A}^{-1} \cdot \boldsymbol{A}\right) \cdot \boldsymbol{s} = \boldsymbol{s}^\dagger \cdot \boldsymbol{s} \\
&= \sum_k s_k^* \cdot s_k = \sum_k |s_k|^2 \\
&= \|\boldsymbol{s}\|_2^2
\end{aligned}$$

Othogonal Transform

- ❑ Othogonal Transforms (what we use in coding)



Orthogonal Matrix

- ■ Special case of unitary matrix: All matrix elements are real values
- ➡ Inverse matrix is equal to the transpose

$$\boldsymbol{A}^{-1} = \boldsymbol{A}^{\mathrm{T}}$$

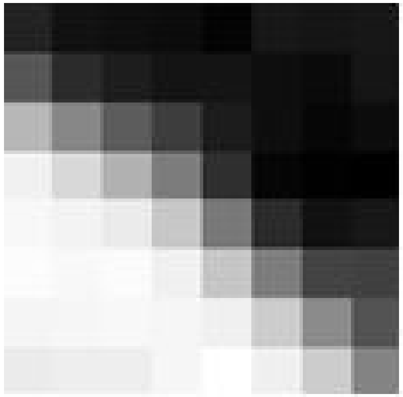
Basis Vectors

- ■ Columns of synthesis matrix $\boldsymbol{B}$
- ➡ Rows of analysis matrix $\boldsymbol{A} = \boldsymbol{B}^{\mathrm{T}}$

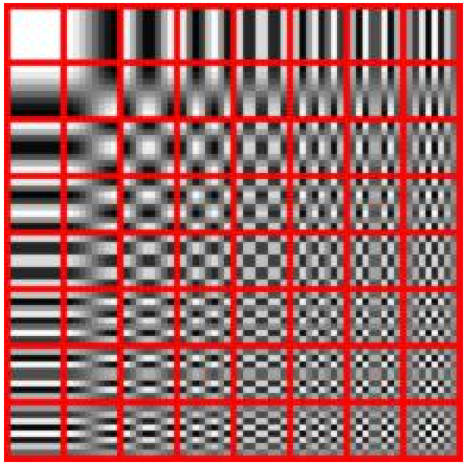
$$\boldsymbol{A} = \begin{bmatrix} \text{——} & \boldsymbol{b}_0 & \text{——} \\ \text{——} & \boldsymbol{b}_1 & \text{——} \\ \text{——} & \boldsymbol{b}_2 & \text{——} \\ & \vdots & \\ \text{——} & \boldsymbol{b}_{N-1} & \text{——} \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} | & | & | & & | \\ \boldsymbol{b}_0 & \boldsymbol{b}_1 & \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_{N-1} \\ | & | & | & & | \end{bmatrix}$$

Transform - Objective

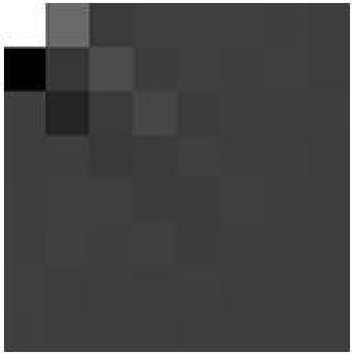
- ❑ Objective: to find alternative representation of the image/signal that is more compact



*

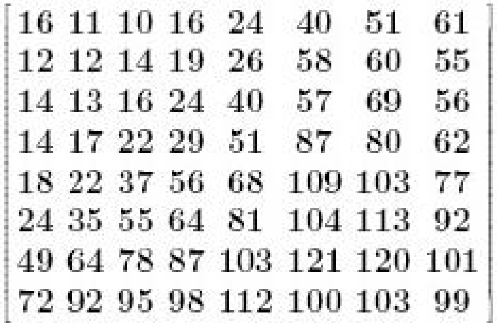


=



*

$$\begin{matrix} 8 & 24 & -2 & 0 & 0 & 0 & 0 & 0 \\ -31 & -4 & 6 & -1 & 0 & 0 & 0 & 0 \\ 0 & -12 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$



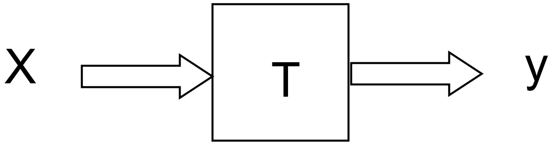
$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120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

Quant Table:

Block Transform

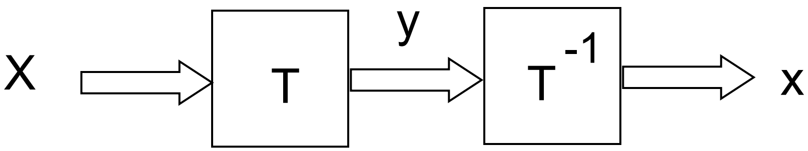
- ❑Divide input data into blocks, encode each block separately
- ❑Matrix representation:

$$\mathbf{y}_{N\times 1} = \mathbf{T}_{N\times N}\mathbf{x}_{N\times 1}$$



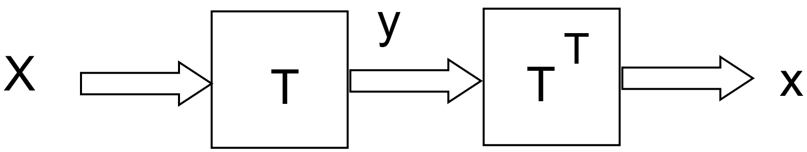
- Inverse Transform:

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{y}$$



- Orthogonal (orthonormal) Transform:

$$\mathbf{T}^{-1} = \mathbf{T}^{T}$$



- Biorthogonal Transform: $\mathbf{T}^{-1} \neq \mathbf{T}^{T}$

Property of Orthogonal Transform

$\mathbf{y} = \mathbf{T}\mathbf{x}$ T orthogonal.

1. Orthogonal transform preserves the energy:

$$\|\mathbf{y}\|^2 = \|\mathbf{x}\|^2$$

Proof:

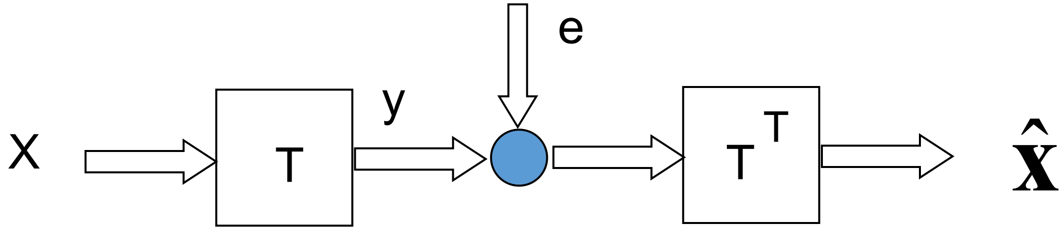
$$\|\mathbf{y}\|^2 = \mathbf{y}^T\mathbf{y} = \mathbf{x}^T\mathbf{T}^T\mathbf{T}\mathbf{x} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|^2$$

Property of Orthogonal Transform

$$\mathbf{y} = \mathbf{T}\mathbf{x}$$ T orthogonal.

2. Orthogonal transform does not amplify the noise:

$$\|\mathbf{e}\|^2 = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$



Proof:

$$\text{Let } \mathbf{v} = \mathbf{T}^T\mathbf{e}, \quad \text{then } \|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{v}\|^2 = \|\mathbf{e}\|^2$$

Degree of Freedom in Orthogonal Transforms

- ❑ Given a set of orthogonal basis A=[a_1, a_2,…, a_k] in R^d, what is the the DoF of matrix A?
  - Not k x d, because $A^TA=I_d$, DoF(A) = kd – (1/2)k(k-1) ? {Stiefle Manifold}

$$DoF\{S(k,d)\} = k \times d - \frac{1}{2}(k-1)$$

  - No, because rotation of the basis should be invariant, i.e, if span(A_1)=span(A_2), then A_1=A_2.
  - The true DoF(A) is characterized by the Grassmann manifold, it is kd – k^2.

$$DoF\{G(k,d)\} = kd - k^2$$

Karhunen-Loéve Transform (KLT)

❑Also known as Hotelling transform, Principle Component method (energy preserving interpretation).

❑Goal: To decorrelate the input with an orthogonal transform:

$$y = Tx,\ R_{yy} = yy^T \quad ➡ \quad yy^T = (Tx)(Tx)^T = T(xx^T)T^T = TR$$

1. We want $\{y_i\}$ uncorrelated $\Rightarrow$ $\mathbf{R_{yy}}$ diagonal.
2. T is orthogonal ➔ $\mathbf{R_{xx}T}^T = \mathbf{T}^T\mathbf{R_{yy}}.$

➔ Rows of T should be the eigenvectors of R_{xx}.
the variances of {yi} should be the eigenvalues of R_{xx}.

Recall: Eigen-decomposition: If A is square, ➔ $\mathbf{A} = \mathbf{UDU}^{-1}.$

D: diagonal matrix with engenvalues on the diagonal.

U: columns are eigenvectors.

A symmetric ➔ U orthonormal ➔ $\mathbf{U}^T\mathbf{AU} = \mathbf{D}$

➔ The KLT transform should be chosen as $\mathbf{T} = \mathbf{U}^T.$

Principal Component Analysis (PCA)

- Given: N data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ in R^d

- We want to find a new set of features that are linear combinations of original ones:

$$u(\mathbf{x}_i) = \mathbf{u}^T(\mathbf{x}_i - \boldsymbol{\mu})$$

- ($\boldsymbol{\mu}$: mean of data points)

- Choose unit vector **u** in R^d that captures the most data variance (max energy preservation)

Principal Component Analysis

- Direction that maximizes the variance of the projected data:

Maximize $$\frac{1}{N}\sum_{i=1}^{N} \underbrace{\mathbf{u}^{\mathrm{T}}(\mathbf{x}_i - \mu)}_{\text{Projection of data point}}(\mathbf{u}^{\mathrm{T}}(\mathbf{x}_i - \mu))^{\mathrm{T}}$$ subject to ||**u**||=1

$$= \quad \mathbf{u}^{\mathrm{T}} \underbrace{\left[1/\mathrm{N} \sum_{i=1}^{N} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^{\mathrm{T}} \right]}_{\text{Covariance matrix of data}} \mathbf{u}$$

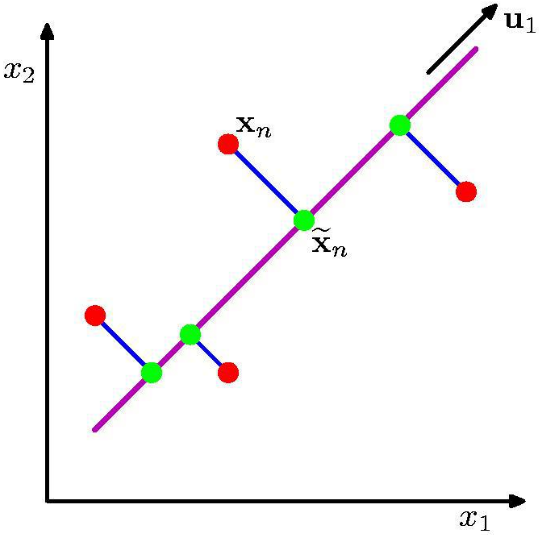
$$= \quad \mathbf{u}^{\mathrm{T}} \Sigma \mathbf{u}$$

The direction that maximizes the variance is the eigenvector associated with the largest eigenvalue of Σ

PCA- Principal Component Analysis

- Formulation:
  - Find projections, that the information/energy of the data are maximally preserved

$$\max_{W} E\{x^T W x\}, \ s.t., \ W^T W = I$$



  - Matlab: [A, s, eigv]=princomp(X);

PCA algorithm

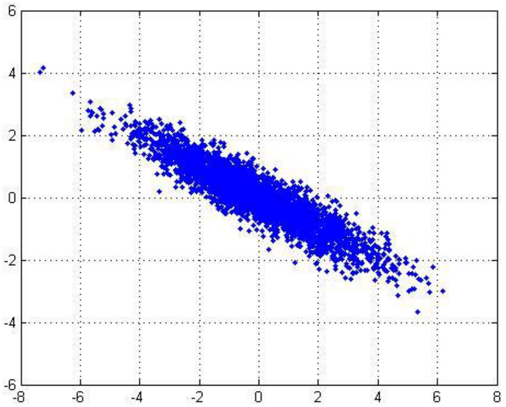
PCA algorithm(**X**, *k*): top *k* eigenvalues/eigenvectors

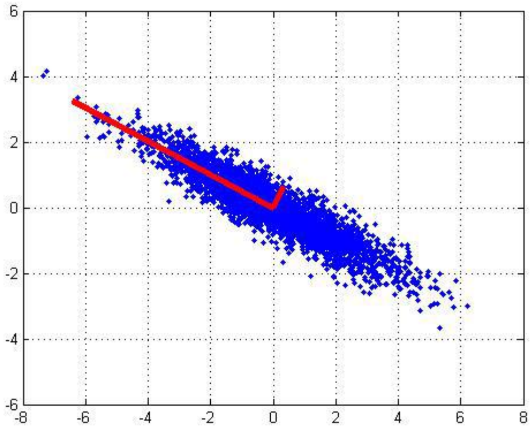
% $\underline{\mathbf{X}}$ = N × m data matrix,
% … each data point $\mathbf{x}_i$ = column vector, i=1..m

- $\underline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i$
- **X** ← subtract mean $\underline{\mathbf{x}}$ from each column vector $\mathbf{x}_i$ in $\underline{\mathbf{X}}$
- **Σ** ← **XX**^{T} … covariance matrix of **X**
- $\{\lambda_i, \mathbf{u}_i\}_{i=1..N}$ = eigenvectors/eigenvalues of Σ
  … $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$
- Return $\{\lambda_i, \mathbf{u}_i\}_{i=1..k}$
  % top *k* principle components

PCA Algorithm

- ❑ Center the data:
  - X = X – repmat(mean(x), [n, 1]);
- ❑Principal component #1 points in the direction of the largest variance
- ❑Each subsequent principal component…
  - is orthogonal to the previous ones, and
  - points in the directions of the largest variance of the residual subspace
- ❑ Solved by finding Eigen Vectors of the Scatter/Covarinace matrix of data:
  - S = cov(X); [A, eigv]=Eig(S)

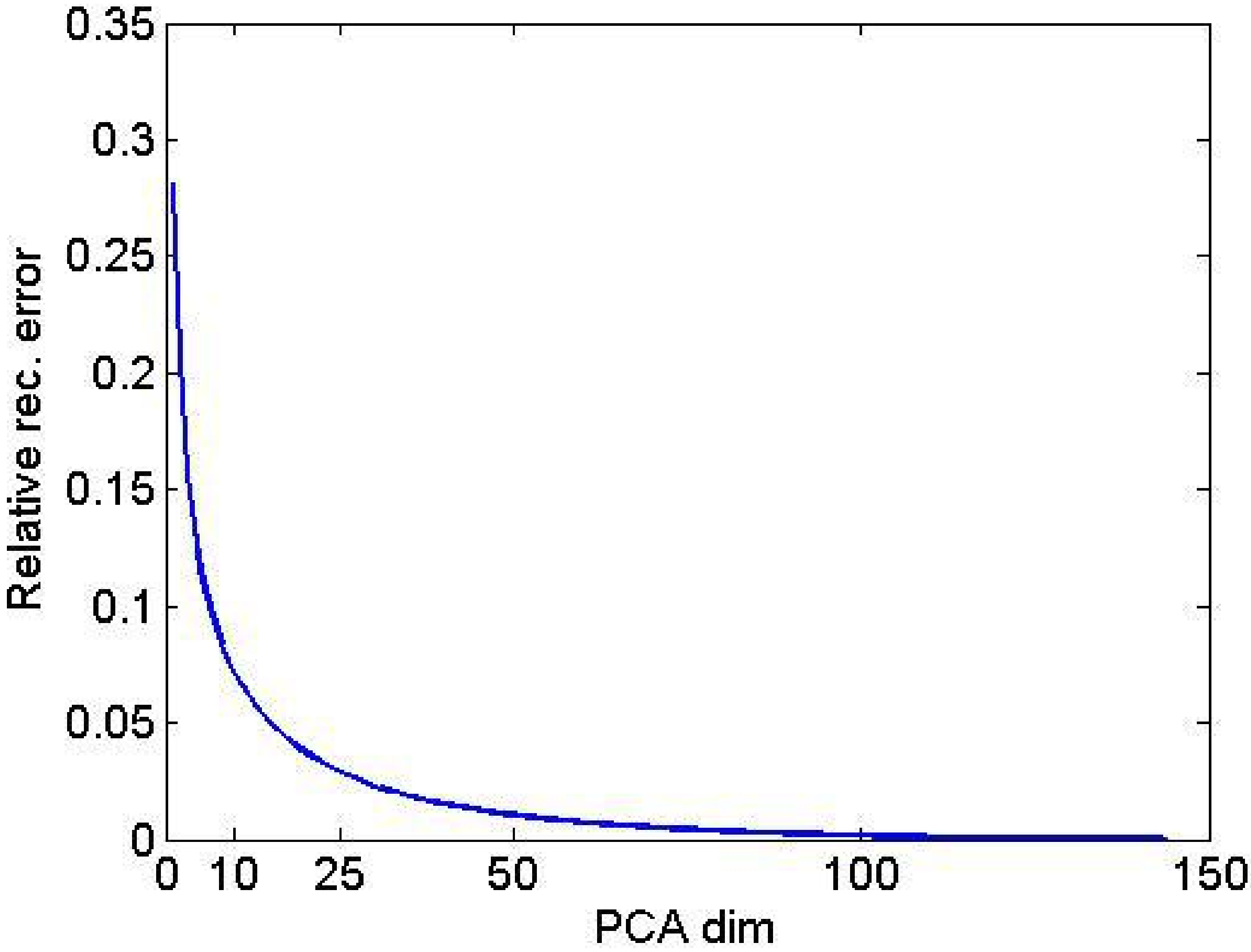




PCA projection of Images

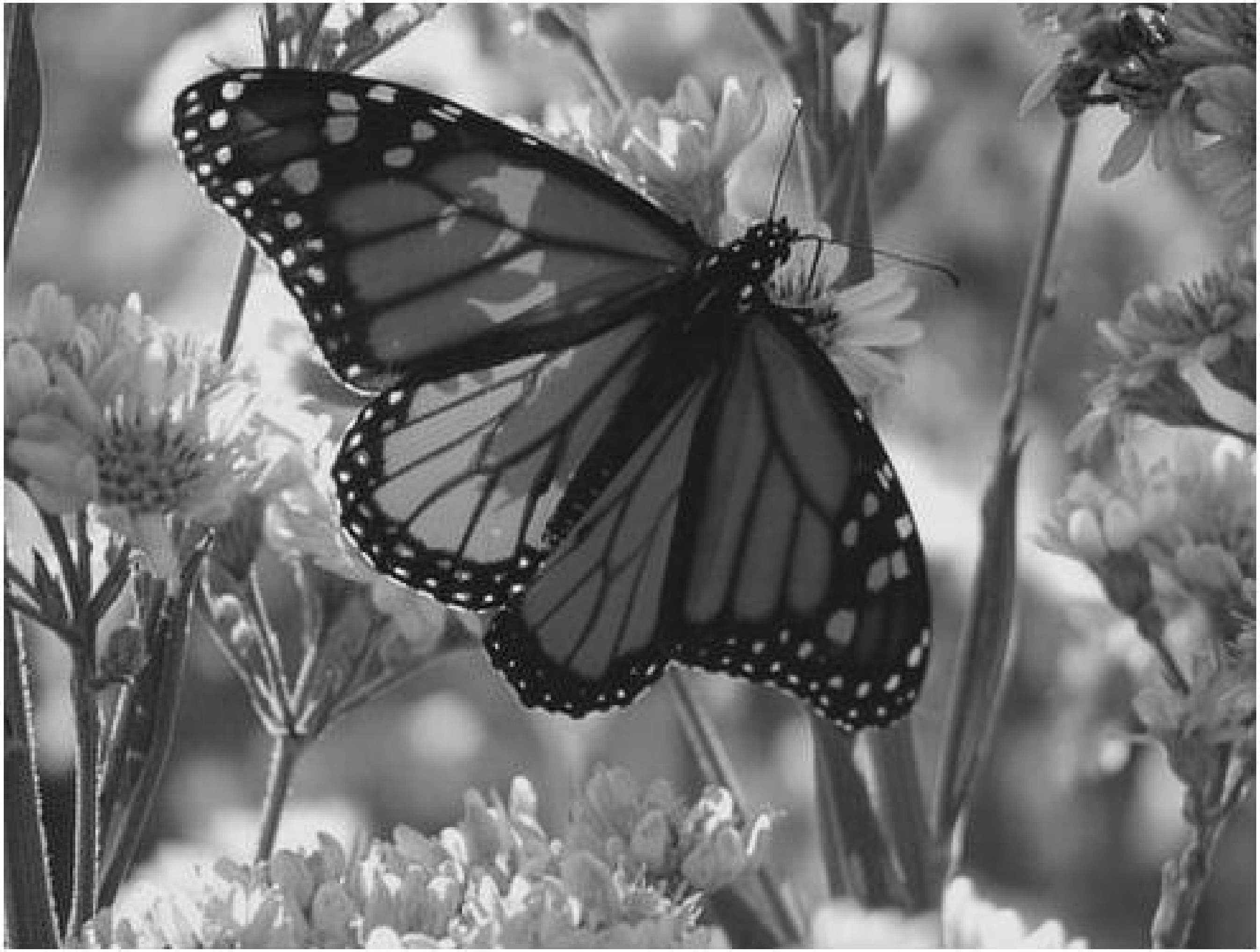
- Divide the original 372x492 image into patches:
  - Each patch is an instance that contains 12x12 pixels on a g
- View each as a 144-D vector

Eigen Values (energy) and PCA dim



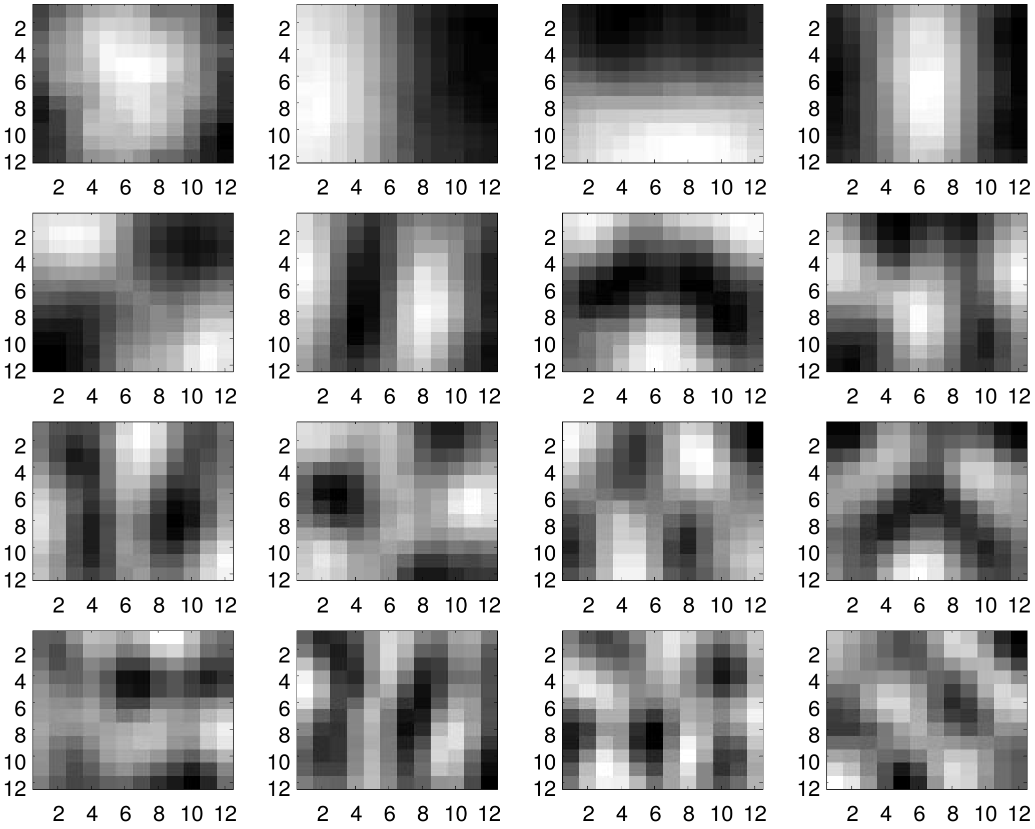
PCA compression: kd=144 >> kd=60

- 12x12 block to 60-D vector



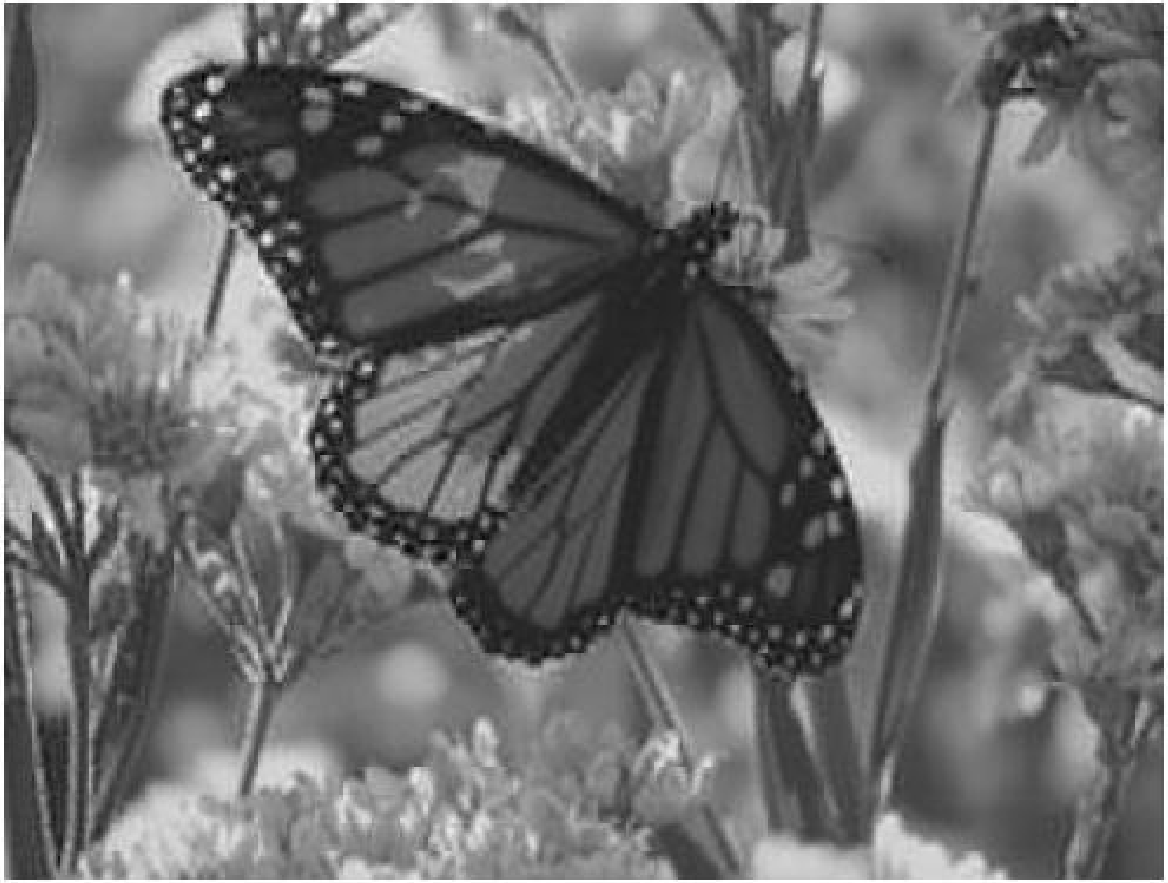
16 most important eigenvectors

- ❑ 16 Most important 12x12 block basis



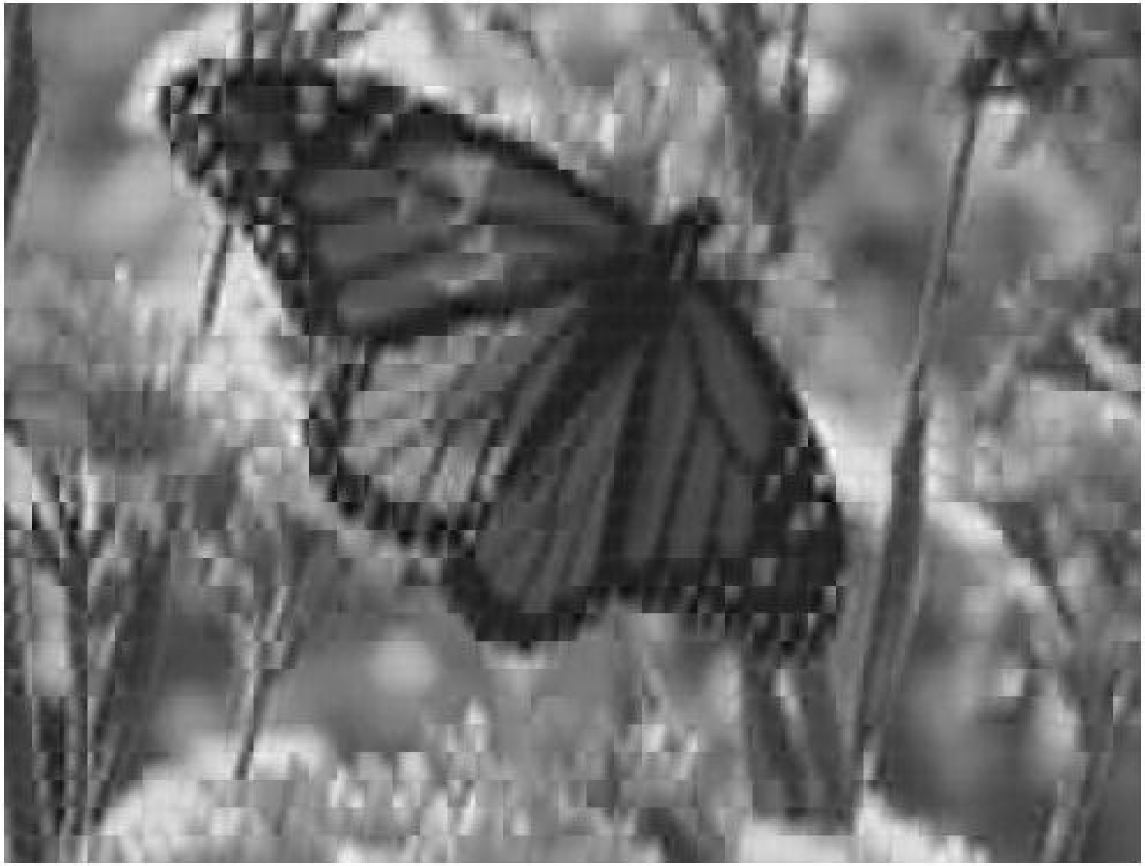
PCA compression: 144d to 16d

- ❑ 12x12 pixel block: 16D vector



PCA compression: 144D >> 6D

- 12x12 pixel block represented as a 6-D vector:

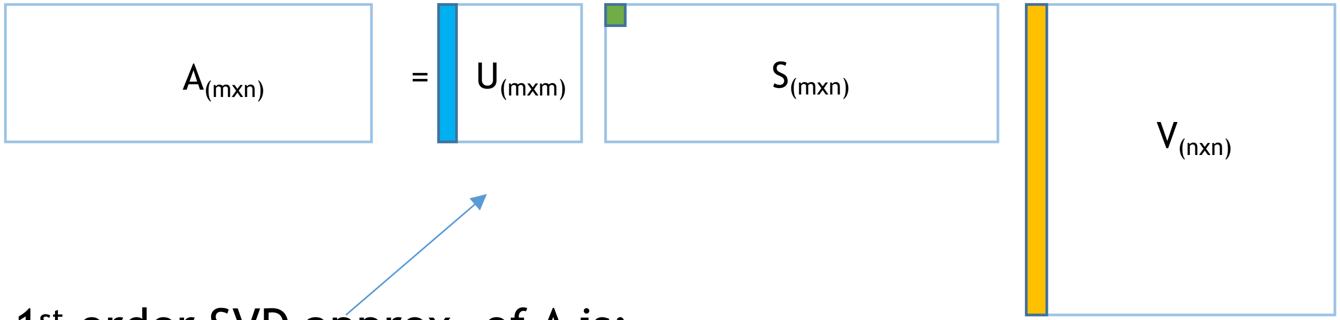


SVD

- ❑ PCA: pull the *n* x *m* image blocks as vector in R^{nxm}.
- ❑The Singular Value Decomposition (SVD) of an nxm matrix A, is,

$$A = USV^T = \sum \sigma_i u_i v_i^t$$

  - Where the diagonal of S are the eigen values of AA^T, $[\sigma_1, \sigma_2, \ldots, \ \sigma_n]$
  - U are eigenvectors of AA^T, and V are eigen vectors of A^TA, the outer product of $u_i v_i^T$, are basis of A in reconstruction:



The 1st order SVD approx. of A is:

$$\sigma_1 * U(:,1) * V(:,1)^T$$

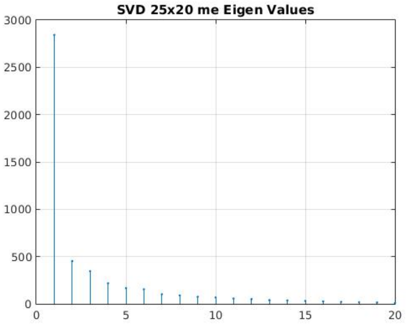
SVD approximation of an image

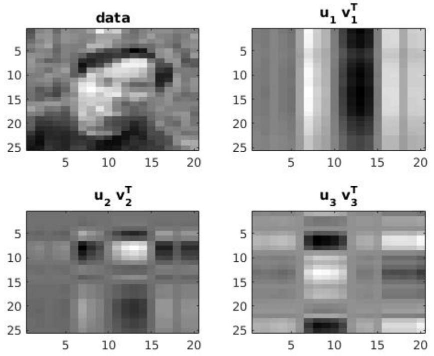
❑ Very easy…

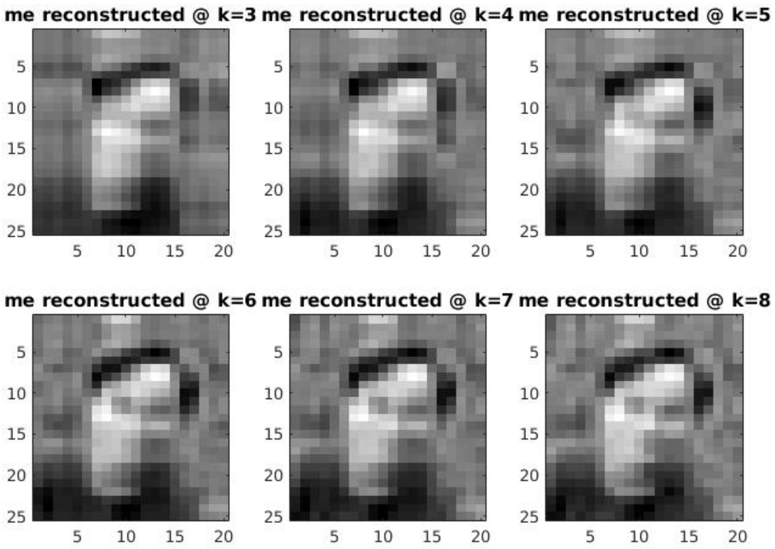
```
function [x]=svd_approx(x0, k)
dbg=0;
if dbg
    x0= fix(100*randn(4,6));
    k=2;
end

[u, s, v]=svd(x0);
[m, n]=size(s);
x = zeros(m, n);
sgm = diag(s);

for j=1:k
    x = x + sgm(j)*u(:,j)*v(:,j)';
end
```

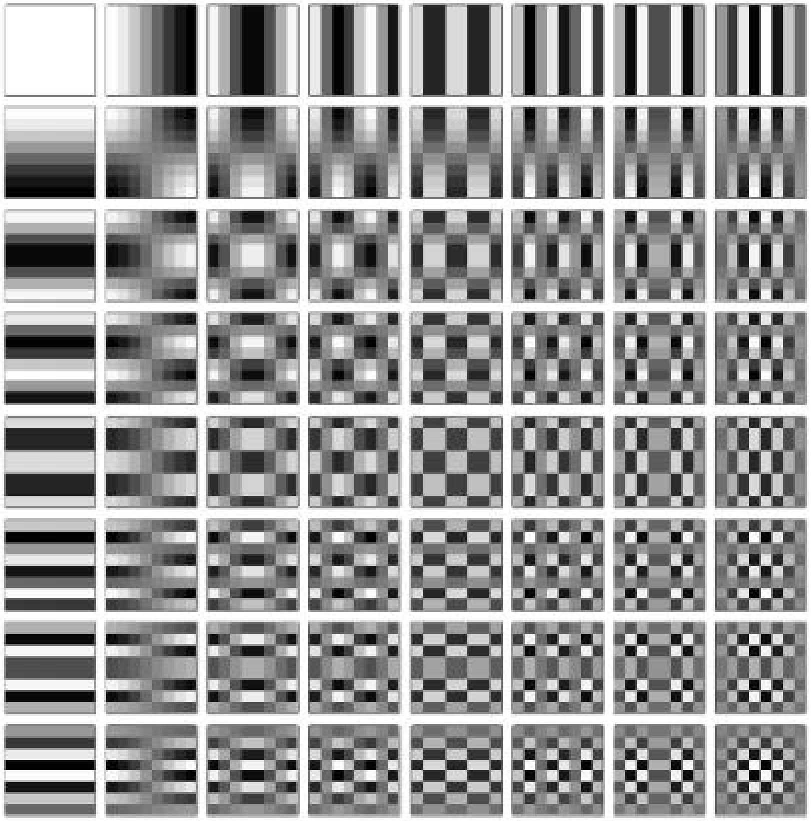






Outline

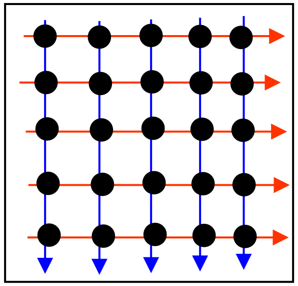
- ❑ Lecture 05 Arithmetic Coding Re-Cap
- ❑ Signal Transform
  - KLT/PCA
  - DCT



8x8 DCT basis

2-D Block Transform

- ❑ Problem with SVD/PCA
  - Data dependent
  - Non-separable Transform
- ❑ Separable approach:
  - Apply transform to each row, then to each column



An N x N block

Matrix form: $\mathbf{Y}_{N\times N} = \underset{\substack{\uparrow \\ \text{col tx}}}{\mathbf{T}_{N\times N}} \mathbf{X}_{N\times N} \underset{\substack{\uparrow \\ \text{row tx}}}{\mathbf{T}^{T}_{N\times N}}$

2-D Transform via Kronecker Product

- Kronecker product: $\mathbf{A} \otimes \mathbf{B}$ Matlab function: kron ( )
  - ❑ The result is a large matrix formed by taking all possible products between the elements of A and those of B.
  - ❑ Example: If A is 2x3 matrix, then

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{1,1}\mathbf{B} & A_{1,2}\mathbf{B} & A_{1,3}\,\mathbf{B} \\ A_{2,1}\mathbf{B} & A_{2,2}\mathbf{B} & A_{2,3}\mathbf{B} \end{bmatrix}$$

❑Note: $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}.$

❑Example:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \qquad \mathbf{B} \otimes \mathbf{A} = \begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}$$

2-D Transform via Kronecker Product

- 2-D *separable transform* can be implemented as 1-D transform via Kronecker product:

Def: $\mathbf{x}_i$: i-th col of X.

$\mathbf{y}_i$: i-th col of Y.

$$\vec{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_1^T & \cdots & \mathbf{x}_N^T \end{bmatrix}^T$$

$$\vec{\mathbf{y}} = \begin{bmatrix} \mathbf{y}_1^T & \cdots & \mathbf{y}_N^T \end{bmatrix}^T$$

then $$\boxed{\mathbf{Y} = \mathbf{T}\mathbf{X}\mathbf{T}^T \Leftrightarrow \vec{\mathbf{y}} = (\mathbf{T} \otimes \mathbf{T})\vec{\mathbf{x}}}$$

- Filtering complexity instead of $O(n^2)$, now O(n).
  - ❑ DCT, SIFT DoG

2-D Separable Transform via Kronecker Product

$$\boxed{\mathbf{Y} = \mathbf{T}\mathbf{X}\mathbf{T}^{T} \Longleftrightarrow \vec{\mathbf{y}} = \left(\mathbf{T} \otimes \mathbf{T}\right)\vec{\mathbf{x}}}$$

Example:

$$\mathbf{T} = \begin{bmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{bmatrix}\begin{bmatrix} a & c \\ b & d \end{bmatrix}\begin{bmatrix} t_{00} & t_{10} \\ t_{01} & t_{11} \end{bmatrix} = \begin{bmatrix} t_{00}a + t_{01}b & t_{00}c + t_{01}d \\ t_{10}a + t_{11}b & t_{10}c + t_{11}d \end{bmatrix}\begin{bmatrix} t_{00} & t_{10} \\ t_{01} & t_{11} \end{bmatrix}$$

$$= \begin{bmatrix} t_{00}t_{00}a + t_{00}t_{01}b + t_{01}t_{00}c + t_{01}t_{01}d & t_{10}t_{00}a + t_{10}t_{01}b + t_{11}t_{00}c + t_{11}t_{01}d \\ t_{00}t_{10}a + t_{00}t_{11}b + t_{01}t_{10}c + t_{01}t_{11}d & t_{10}t_{10}a + t_{10}t_{11}b + t_{11}t_{10}c + t_{11}t_{11}d \end{bmatrix}$$

$$\vec{\mathbf{y}} = \begin{bmatrix} y_{00} \\ y_{10} \\ y_{01} \\ y_{11} \end{bmatrix} = \begin{bmatrix} t_{00}t_{00} & t_{00}t_{01} & t_{01}t_{00} & t_{01}t_{01} \\ t_{00}t_{10} & t_{00}t_{11} & t_{01}t_{10} & t_{01}t_{11} \\ t_{10}t_{00} & t_{10}t_{01} & t_{11}t_{00} & t_{11}t_{01} \\ t_{10}t_{10} & t_{10}t_{11} & t_{11}t_{10} & t_{11}t_{11} \end{bmatrix}\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} t_{00}\mathbf{T} & t_{01}\mathbf{T} \\ t_{10}\mathbf{T} & t_{11}\mathbf{T} \end{bmatrix}\vec{\mathbf{x}} = \left(\mathbf{T} \otimes \mathbf{T}\right)\vec{\mathbf{x}}$$

Discrete Cosine Transform (DCT)

❑DCT is the approximation of the KLT of AR(1) signal when its correlation coefficient ρ is close to 1 (e.g. 0.95)

- Definition:

$$\mathbf{C}_{i,j} = a\cos\left(\frac{(2j+1)\,i\,\pi}{2N}\right), \quad \mathrm{i,\,j = 0, \ldots, N\text{-}1.}$$

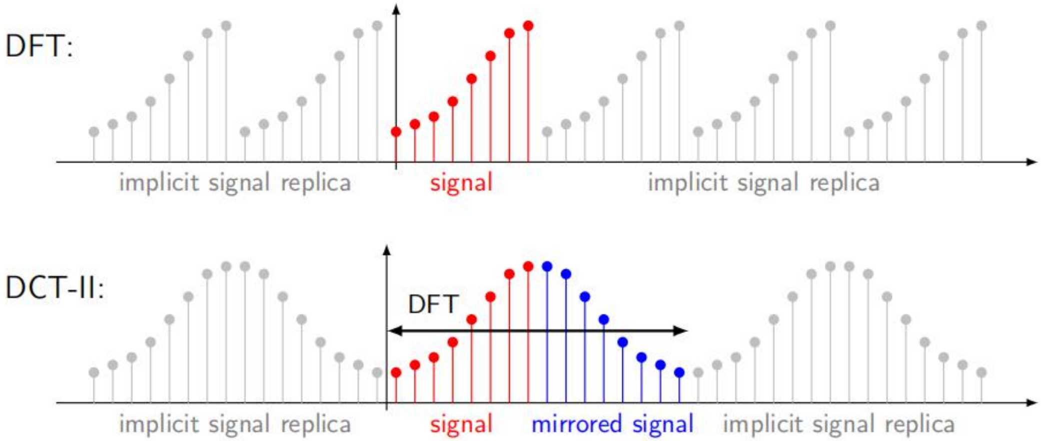
$$a = \sqrt{1/N} \quad \mathrm{for\ i = 0,}$$

$$a = \sqrt{2/N} \quad \mathrm{for\ i = 1, \ldots, N\text{-}1.}$$

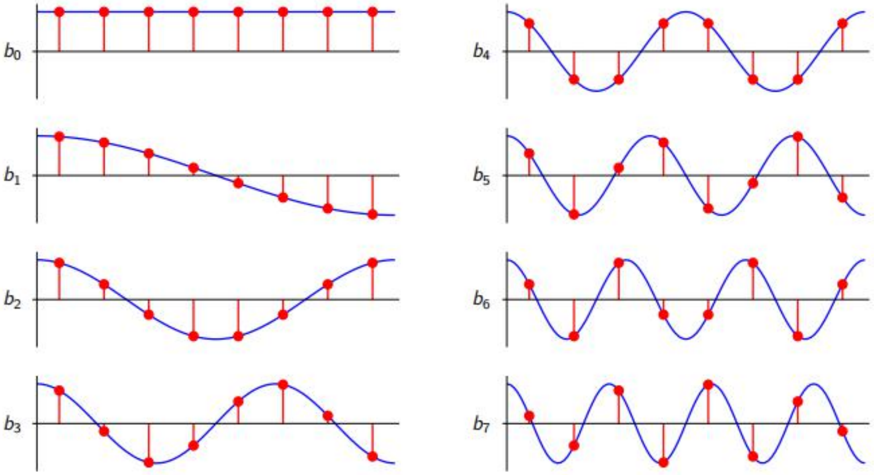
- Matlab function:
  - ❑`dct(eye(N));`

DCT implemented as DFT

- ❑ Implicit mirroring input signal to have real number FFT



- ❑8-point DCT Basis



DCT

- ❑ Definition:

$$\mathbf{C}_{i,j} = a \cos\left(\frac{(2j+1)\,i\,\pi}{2N}\right), \quad \mathrm{i, j = 0, \ldots, N-1.}$$

$$a = \sqrt{1/N} \quad \text{for i = 0,}$$

$$a = \sqrt{2/N} \quad \text{for i = 1, ..., N-1.}$$

- N = 2: DCT=Haar Transform: $\mathbf{C}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

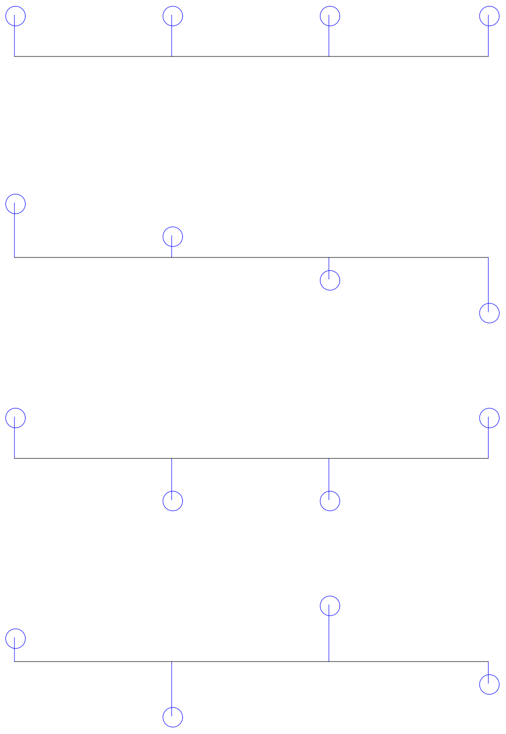
$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \mathbf{C}_2 \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} x_0 + x_1 \\ x_1 - x_1 \end{bmatrix}$$

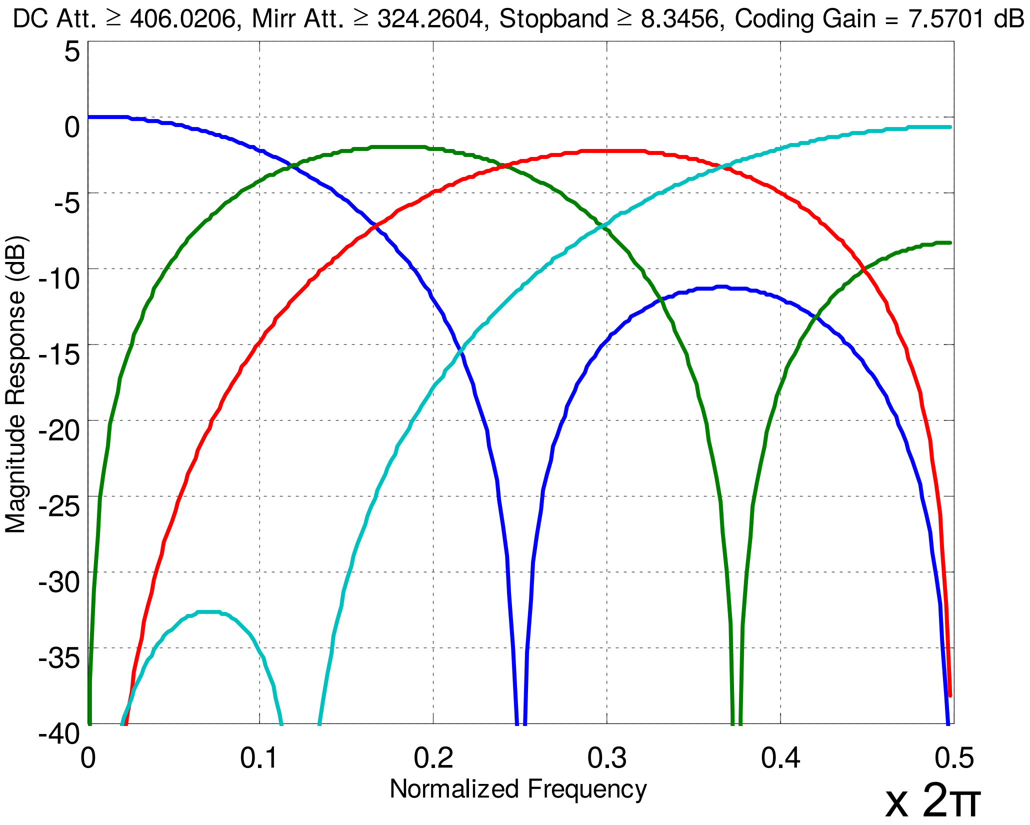
- y0 captures the mean of x0 and x1 (low-pass)
  - ❑ x0 = x1 = 1 ➔ y0 = sqrt(2) (DC), y1 = 0
- y1 captures the difference of x0 and x1 (high-pass)
  - ❑ x0 = 1, x1 = -1 ➔ y0 = 0 (DC), y1 = sqrt(2).

4-point DCT Freq Response

- ❑Four subbands
- ❑2 sym. filters
- ❑2 anti-sym.

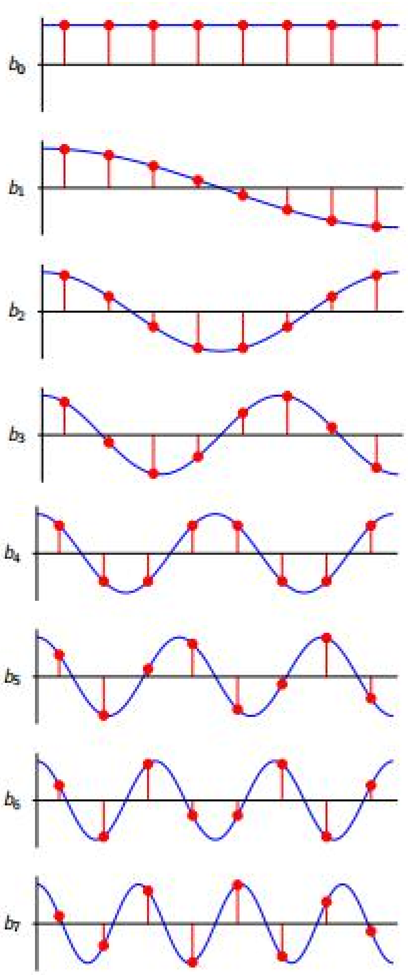
0.5000	0.5000	0.5000	0.5000
0.6533	0.2706	-0.2706	-0.6533
0.5000	-0.5000	-0.5000	0.5000
0.2706	-0.6533	0.6533	-0.2706

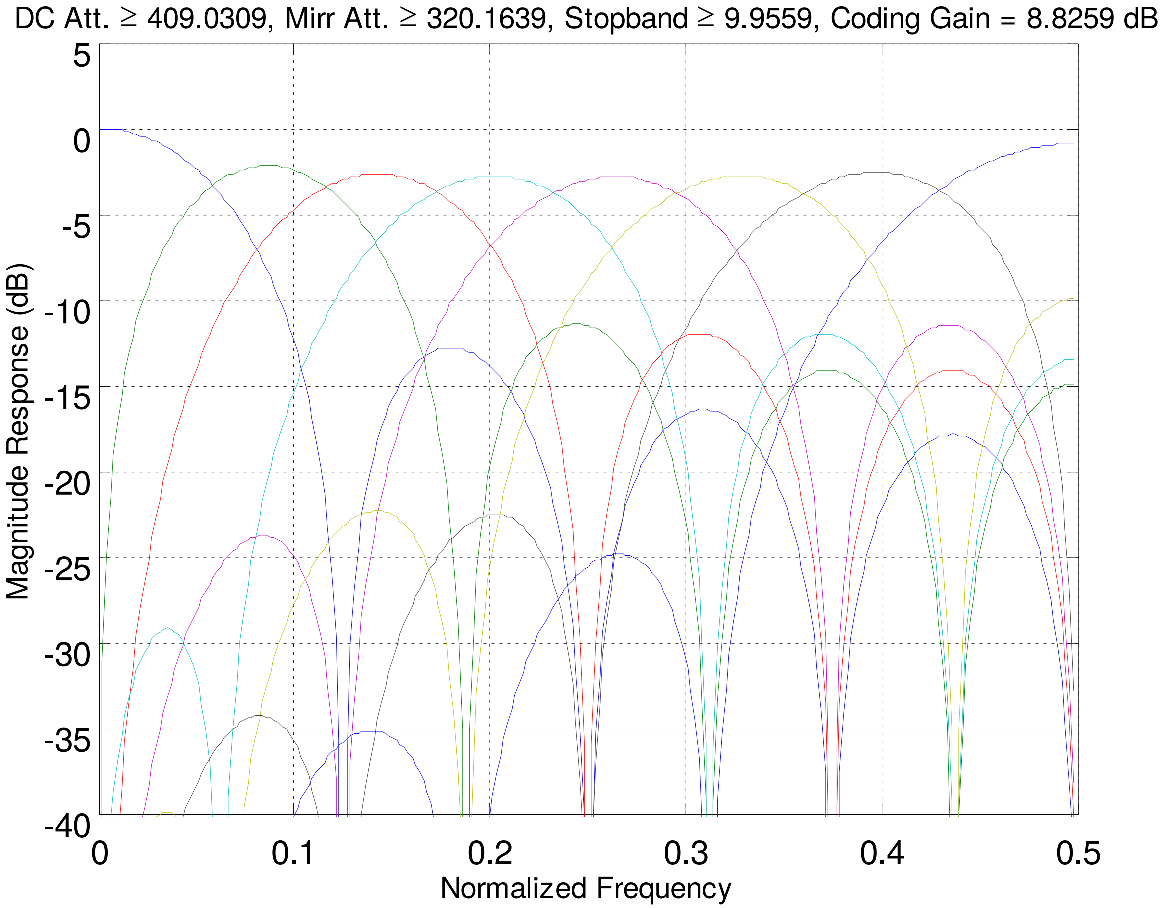




8-point DCT

- ❑Eight subbands:4 sym., 4 anti-sym.





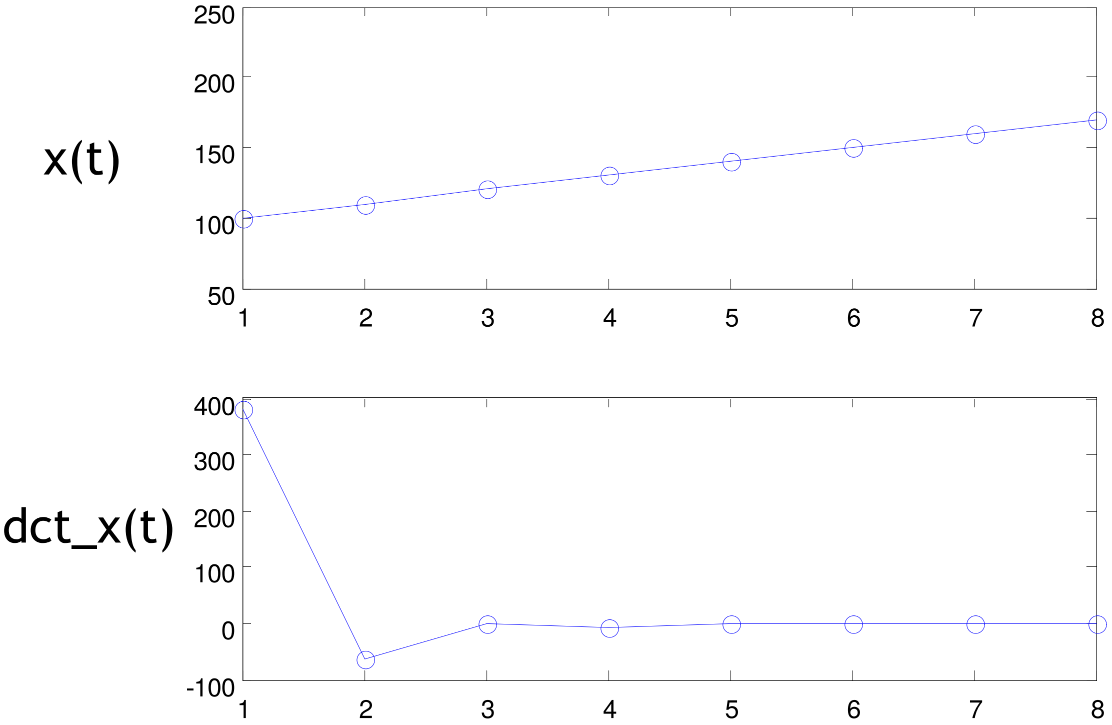
x 2π

Example: 1-D DCT

- x = [100 110 120 130 140 150 160 170]^T;
- 8-point DCT:

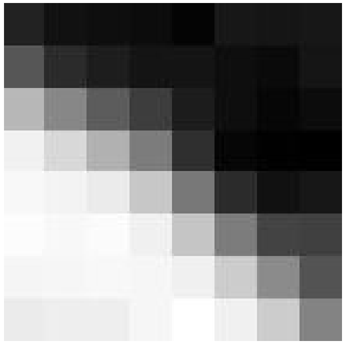
[381.8377, -64.4232, 0.0, -6.7345, 0.0, -2.0090, 0.0, -0.5070]

Most energy are in the first 2 coefficients.



2-D 8-point DCT Example

- Original Data:



89	78	76	75	70	82	81	82
122	95	86	80	80	76	74	81
184	153	126	106	85	76	71	75
221	205	180	146	97	71	68	67
225	222	217	194	144	95	78	82
228	225	227	220	193	146	110	108
223	224	225	224	220	197	156	120
217	219	219	224	230	220	197	151

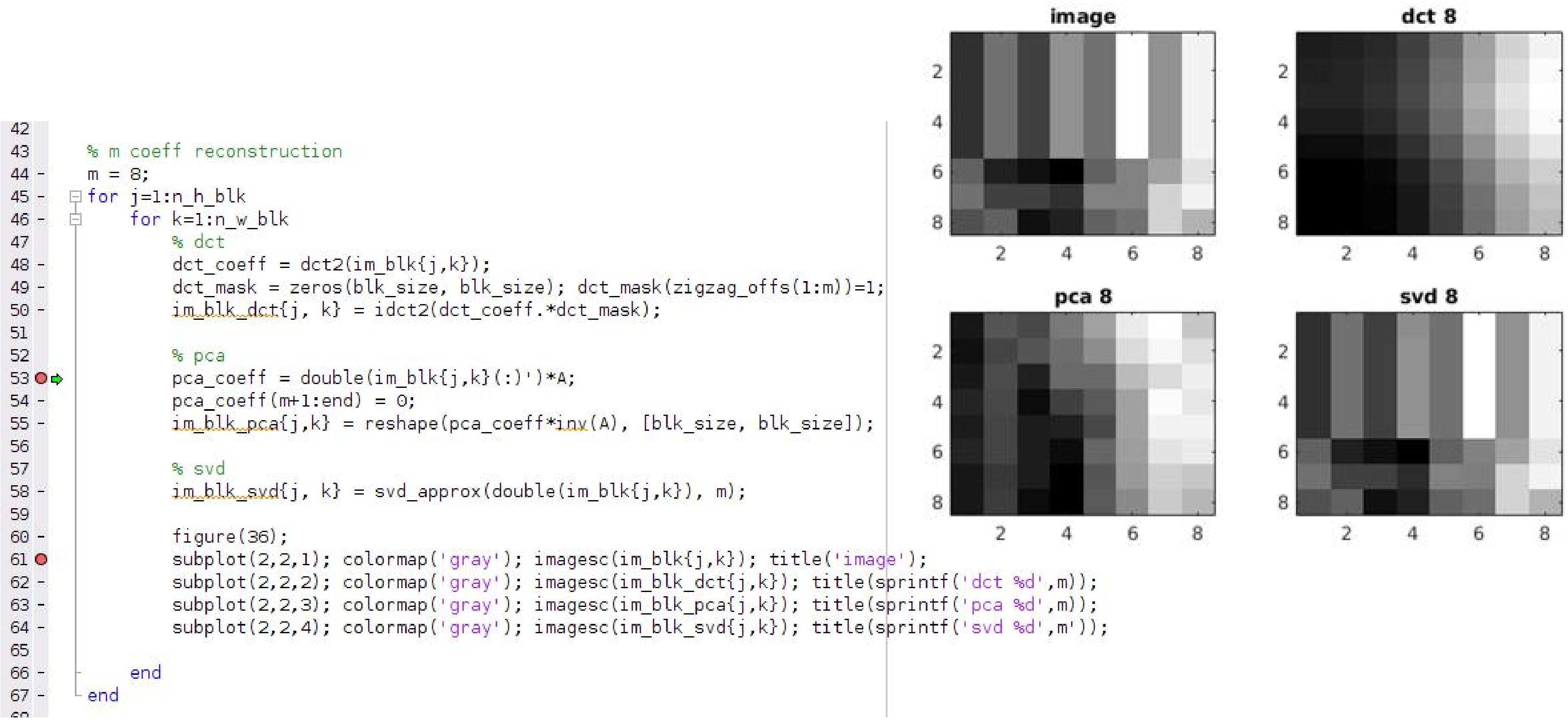
- 2-D DCT Coefficients (after rounding to integers):

1155	259	-23	6	11	7	3	0
-377	-50	85	-10	10	4	7	-3
-4	-158	-24	42	-15	1	0	1
-2	3	-34	-19	9	-5	4	-1
1	9	6	-15	-10	6	-5	-1
3	13	3	6	-9	2	0	-3
8	-2	4	-1	3	-1	0	-2
2	0	-3	2	-2	0	0	-1

Most energy is in the upper-left corner

Matlab Exercise: SVD, PCA, and DCT approximation

- ❑ For the lena image, process it as 8x8 blocks
- ❑ Compute m-coefficients reconstructions



```
% m coeff reconstruction
m = 8;
for j=1:n_h_blk
    for k=1:n_w_blk
        % dct
        dct_coeff = dct2(im_blk{j,k});
        dct_mask = zeros(blk_size, blk_size); dct_mask(zigzag_offs(1:m))=1;
        im_blk_dct{j, k} = idct2(dct_coeff.*dct_mask);

        % pca
        pca_coeff = double(im_blk{j,k}(:)')*A;
        pca_coeff(m+1:end) = 0;
        im_blk_pca{j,k} = reshape(pca_coeff*inv(A), [blk_size, blk_size]);

        % svd
        im_blk_svd{j, k} = svd_approx(double(im_blk{j,k}), m);

        figure(36);
        subplot(2,2,1); colormap('gray'); imagesc(im_blk{j,k}); title('image');
        subplot(2,2,2); colormap('gray'); imagesc(im_blk_dct{j,k}); title(sprintf('dct %d',m));
        subplot(2,2,3); colormap('gray'); imagesc(im_blk_pca{j,k}); title(sprintf('pca %d',m));
        subplot(2,2,4); colormap('gray'); imagesc(im_blk_svd{j,k}); title(sprintf('svd %d',m'));

    end
end
```

Summary

❑ Transforms

- Transform decorrelates pixels and allows for effective quantization in reducing signalling rate
- KL Transform/PCA: de-correlation and energy preserving formulations
- SVD: dealing with n x m tensorial data approximation
- DCT: separable transform, much faster, good de-correlation and energy preserving property