Transform Coding

by Erol Seke

For the course “**Data Compression**”

ESKİŞEHİR OSMANGAZİ UNIVERSITY

What we mean by Orthogonal Linear Transform

Assume that x is a sequence of numbers, presumably with high correlation within.

$$x_n = x_0, x_1, \cdots, x_{N-1}, \quad n = 0, 1, \cdots, N-1$$

than, an orthogonal linear transform can be written as.

$$y_k = \sum_{n=0}^{N-1} f_{kn} x_n \quad , k = 0, 1, \cdots, N-1$$

where f_{kn} is called the transform kernel and for $i \neq j$

$$\sum_{n=0}^{N-1} f_{in} f_{jn} = 0 \quad \text{(orthogonality)}$$

f_{kn} spans the N dimensional space $\mathbb{R}^N$, that is …

…any possible set $\{x_0, x_1, \cdots, x_{N-1}\}$ can be constructed by a weighted sum of f_{kn} as

$$x_k = \sum_{n=0}^{N-1} a_n f_{kn}$$ where a_n are the weights

In matrix form

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \qquad F = \begin{bmatrix} f_{0,0} & f_{0,1} & \cdots & f_{0,N-1} \\ f_{1,0} & f_{1,1} & \cdots & f_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N-1,0} & f_{N-1,1} & \cdots & f_{N-1,N-1} \end{bmatrix}$$

$$Y = FX$$

For orthogonality and $\mathbb{R}^N$ spanning presumptions to hold, $rank(F) = N$

Inverse

If F^{-1} is possible (it should be because $rank(F) = N$)

$G = F^{-1}$ is called inverse of the transform kernel, and

$X = GY$ is called the inverse transform

$$x_n = \sum_{k=0}^{N-1} g_{nk} y_k$$

$$y_k = \sum_{n=0}^{N-1} f_{kn} x_n$$ Forward orthogonal linear transform

$$x_n = \sum_{k=0}^{N-1} g_{nk} y_k$$ Inverse transform

2D

For 2D datasets, the same properties must hold

$$y_{k,l} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x_{m,n} f_{kn,lm}$$

If the above summations can be written as

$$y_{k,l} = \sum_{m=0}^{N-1} \left(\sum_{n=0}^{N-1} x_{m,n} u_{k,n} \right) v_{l,m}$$

the transform is called separable

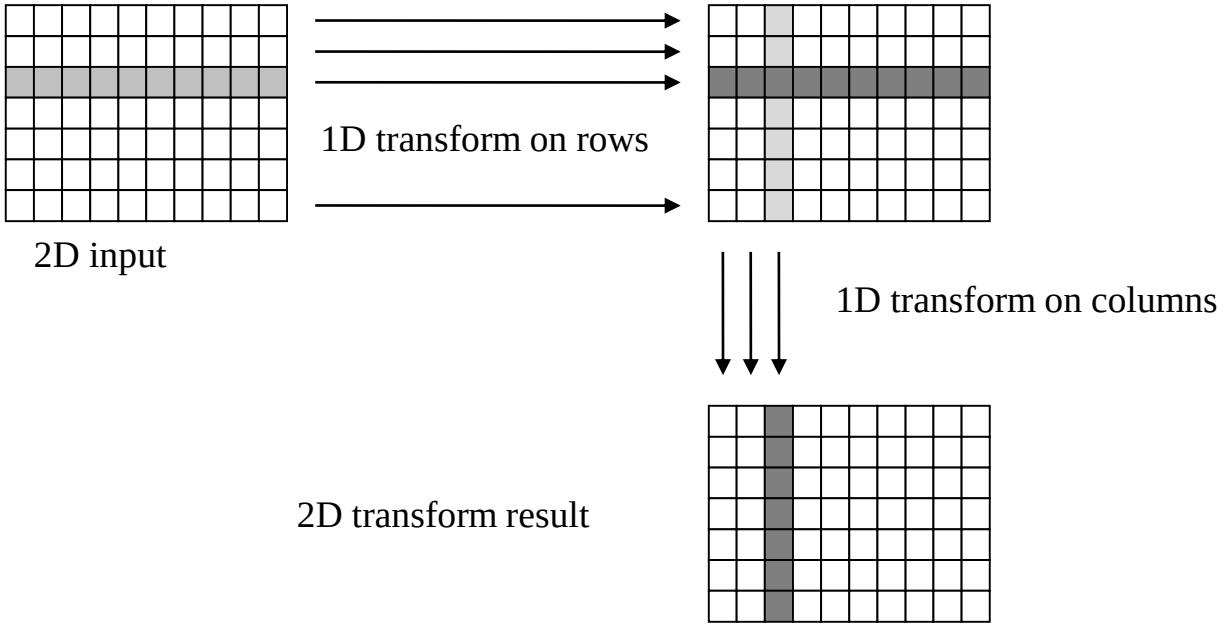
If $u_{i,j} = v_{i,j}$, then the transform is called symmetric

Separable and Symmetric

If the transform is both separable and symmetric, then

The same 1D transform can be applied on the rows of the input data, followed by the application on the columns of the resulting data

to calculate a 2D transform



Discrete Fourier Transform

1D

$$y_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi\left(\frac{kn}{N}\right)}$$

2D

$$y_{k,l} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{m,n} e^{-j2\pi\left(\frac{kn}{N}+\frac{lm}{M}\right)}$$

Inverse 1D

$$x_n = \sum_{k=0}^{N-1} y_k e^{j2\pi\left(\frac{kn}{N}\right)}$$

Inverse 2D

$$x_{m,n} = \sum_{l=0}^{M-1} \sum_{k=0}^{N-1} y_{k,l} e^{j2\pi\left(\frac{kn}{N}+\frac{lm}{M}\right)}$$

What we want from a Transform

to be useful in data compression applications

1. Simplicity in implementation
2. Availability of fast calculation algorithms
3. Possibility of implementation via hardware
4. Usefulness in separating redundant data

We want the transform to be simple enough to be implemented easily (not difficult at least)

For real time applications, the transform must be performed in real time. Read this as “low complexity”.

Hardware implementation is a huge plus, for real time applications.

It should present a reason for employing the transform in compression.
We would not use it just because it is implementable & fast & has inverse.

Discrete Cosine Transform

There exists at least 5 different formulations for DCT

Below is the one used in JPEG compression

2D DCT

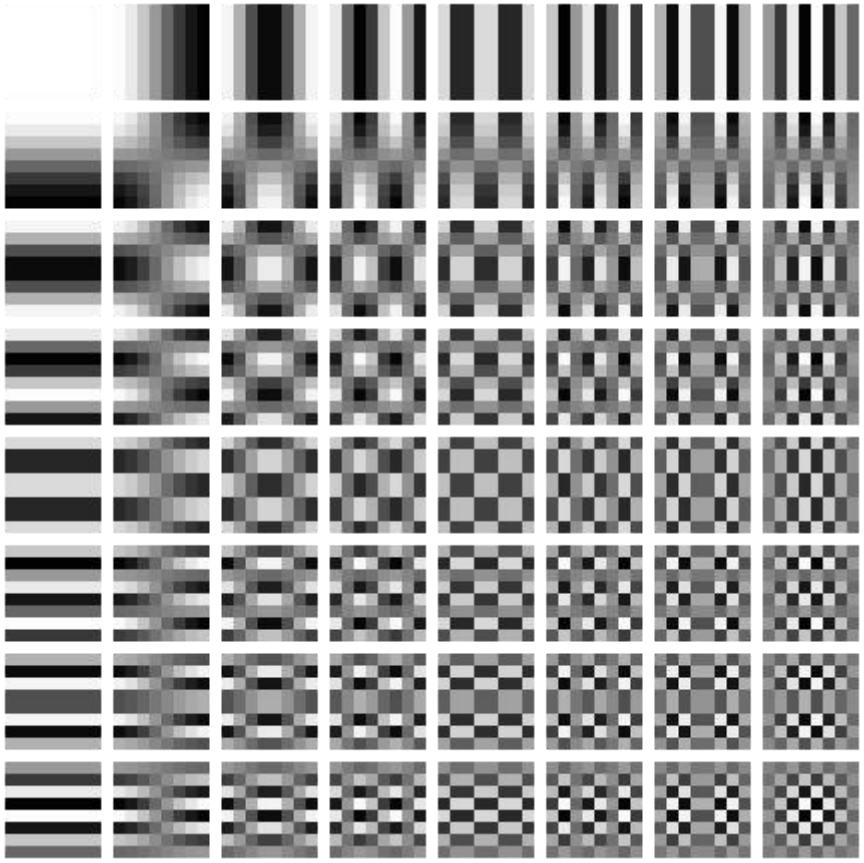
$$y_{k,l} = \sum_{m=0}^{M-1} \alpha(m) \left(\sum_{n=0}^{N-1} \alpha(n) x_{m,n} cos\big((2n+1)k\,\pi/2\,N\big) \right) cos\big((2m+1)l\,\pi/2\,M\big)$$

$$\alpha(s) = \begin{cases} \frac{1}{\sqrt{N}} & , s = 0 \\ \frac{2}{\sqrt{N}} & s \neq 0 \end{cases}$$

1D DCT

$$y_k = \sum_{n=0}^{N-1} x_n cos\left(\left(n + \frac{1}{2} \right) k\,\pi/N \right)$$

2D DCT Kernel

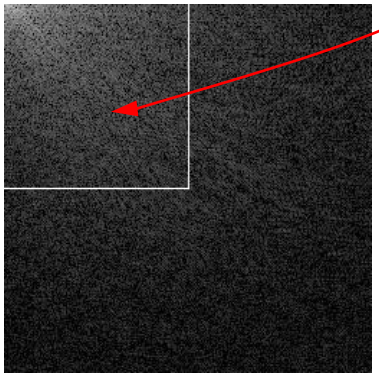


What we want from a Transform

Original image

I

DCT coefficients image



$log(C\{I\}+1)$ scaled

Inverse transform of this part
The rest are assumed zero

$C^{-1}\{Ix\}$

DCT collects energy towards lower frequency components.
Higher frequency components represent detail and edges.
When HF are removed we loose sharpness and get a blurred image

END