Image Transforms

Qiufeng Wang

Qiufeng.Wang@xjtlu.edu.cn

Introduction

- In this lesson we will cover the following topics:
  - Spatial domain vs. transform domain
  - Eigen-Faces
  - Discrete Fourier Transform (DFT)
  - Discrete Cosine Transform (DCT)

Introductory Examples
--**Basis** theory

- Digital basis
  - 110 Binary scale
  - 10 **Decimal scale**
  - A Hexadecimal scale
- Vector basis
  - Polar coordinates (ρ,θ)
  - **Euclidean coordinates** (x,y)
  - (10,10) = **10*(1,0)+10*(0,1)=**10*(1,1)+0*(1,-1)
- Matrix basis

Linear Algebra

Spatial domain vs. transform domain

- Spatial domain

$$\begin{bmatrix} 120 & 72 \\ 40 & 24 \end{bmatrix} = \begin{bmatrix} 120 & \\ & \end{bmatrix} + \begin{bmatrix} & 72 \\ & \end{bmatrix} + \begin{bmatrix} & \\ & 24 \end{bmatrix} + \begin{bmatrix} & \\ 40 & \end{bmatrix}$$

  - In spatial domain the image is "regarded" as a set of pixels each with an intensity; the intensity is a function $x\ (n, m)$ of the position (n, m)
  - What is the basis here?

Spatial domain vs. transform domain

- Transform domain

$$\begin{array}{|c|c|}\hline 120 & 72 \\ \hline 40 & 24 \\ \hline \end{array} = 64\times \begin{array}{|c|c|}\hline +1 & +1 \\ \hline +1 & +1 \\ \hline \end{array} + 32\times \begin{array}{|c|c|}\hline +1 & +1 \\ \hline -1 & -1 \\ \hline \end{array} + 16\times \begin{array}{|c|c|}\hline +1 & -1 \\ \hline +1 & -1 \\ \hline \end{array} + 8\times \begin{array}{|c|c|}\hline +1 & -1 \\ \hline -1 & +1 \\ \hline \end{array}$$

  - In transform domain the image is "regarded" as a combination of some other "images".

Spatial domain vs. transform domain

- Transform domain

$$\begin{bmatrix} 120 & 72 \\ 40 & 24 \end{bmatrix} = 64\times \begin{bmatrix} +1 & +1 \\ +1 & +1 \end{bmatrix} + 32\times \begin{bmatrix} +1 & +1 \\ -1 & -1 \end{bmatrix} + 16\times \begin{bmatrix} +1 & -1 \\ +1 & -1 \end{bmatrix} + 8\times \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

Transformed image

$$\begin{bmatrix} 64 & 16 \\ 32 & 8 \end{bmatrix}$$

Spatial domain vs. transform domain

- Transform domain (forward transform-analysis)

$$\begin{bmatrix} 120 & 72 \\ 40 & 24 \end{bmatrix} = ? \times \begin{bmatrix} +1 & +1 \\ +1 & +1 \end{bmatrix} + ? \times \begin{bmatrix} +1 & +1 \\ -1 & -1 \end{bmatrix} + ? \times \begin{bmatrix} +1 & -1 \\ +1 & -1 \end{bmatrix} + ? \times \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

$$\begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

Spatial domain vs. transform domain

- Transform domain (inverse transform-synthesis)

$$\begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} = 64\times \begin{bmatrix} +1 & +1 \\ +1 & +1 \end{bmatrix} + 32\times \begin{bmatrix} +1 & +1 \\ -1 & -1 \end{bmatrix} + 16\times \begin{bmatrix} +1 & -1 \\ +1 & -1 \end{bmatrix} + 8\times \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

$$\begin{bmatrix} 64 & 16 \\ 32 & 8 \end{bmatrix}$$

Spatial domain vs. transform domain

- Transform domain

$$\begin{bmatrix} 120 & 72 \\ 40 & 24 \end{bmatrix} \Rightarrow \begin{bmatrix} 64 & 16 \\ 32 & 8 \end{bmatrix}$$

- Spatial domain

$$\begin{bmatrix} 120 & 72 \\ 40 & 24 \end{bmatrix} = \begin{bmatrix} 120 & \\ & \end{bmatrix} + \begin{bmatrix} & 72 \\ & \end{bmatrix} + \begin{bmatrix} & \\ & 24 \end{bmatrix} + \begin{bmatrix} & \\ 40 & \end{bmatrix}$$

Spatial domain vs. transform domain

- Transform domain

$$\begin{bmatrix} 64 & 0 \\ 0 & 0 \end{bmatrix} = \overbrace{16\times \begin{bmatrix} +1 & +1 \\ +1 & +1 \end{bmatrix} + 16\times \begin{bmatrix} +1 & +1 \\ -1 & -1 \end{bmatrix} + 16\times \begin{bmatrix} +1 & -1 \\ +1 & -1 \end{bmatrix} + 16\times \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}}^{\text{These images allow us to describe other } 2\times 2 \text{ images}}$$

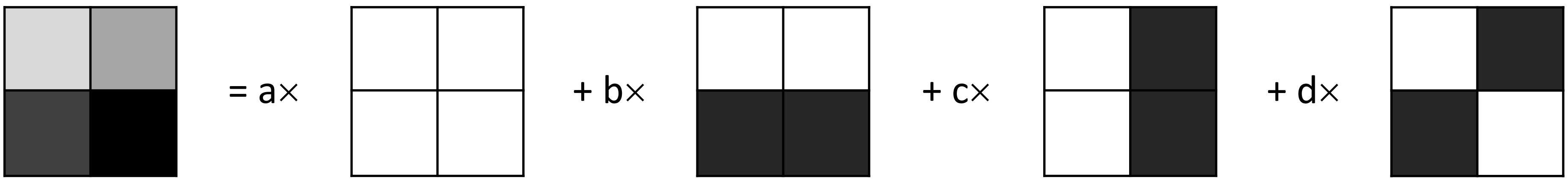
- Spatial domain

$$\begin{bmatrix} 120 & 72 \\ 40 & 24 \end{bmatrix} = \begin{bmatrix} 120 & \\ & \end{bmatrix} + \begin{bmatrix} & 72 \\ & \end{bmatrix} + \begin{bmatrix} & \\ & 24 \end{bmatrix} + \begin{bmatrix} & \\ 40 & \end{bmatrix}$$

Two basic factors of the transform domain

Spatial domain vs. transform domain

- The basis of the transform domain

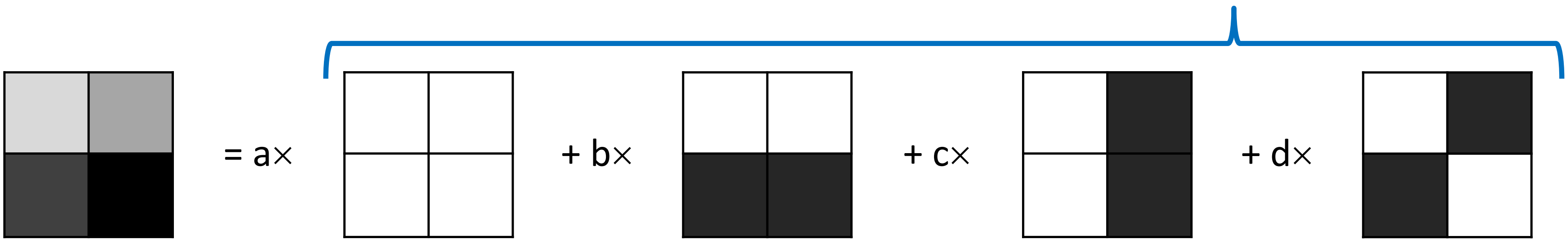


  - To have an image transform you need to have:
    - a basis for the image space; Normally, the number of images of the basis is equal to the number of pixels in the images
    - The basis allows to describe any image of the image space (completed)
    - in general an inverse transform (i.e., the forward transform is invertible)

Spatial domain vs. transform domain

- Coefficients

These images allow us to describe any 2×2 image



  – In transform domain the image is “regarded” as a combination of other “images”.

Why do we need transforms ?

Why do we need transforms ?

- It gives us more flexibility to manipulate data (e.g., filtering, compression)

- Provide us more tools to understand the data (e.g., classification).

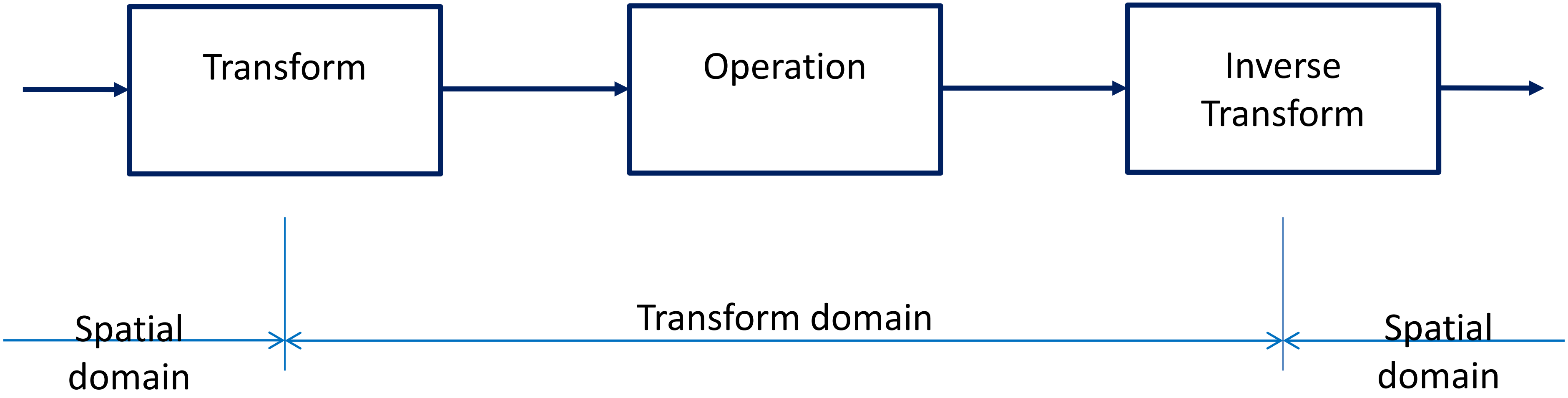
  – Different applications require different types of transform.

Image Transforms

- Many image processing tasks are best performed in the transform domain rather than spatial domain.

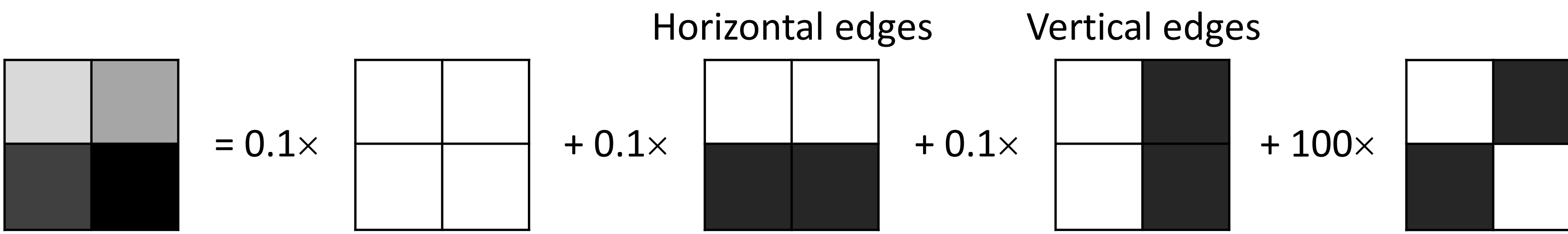
Image Transforms

- Key steps of image transform operation:
  - Transform the image
  - Carry the task(s) in the *transformed domain*.
  - Depending on the end-user application *inverse transform* to return to the spatial domain.



Why do we need transforms ?

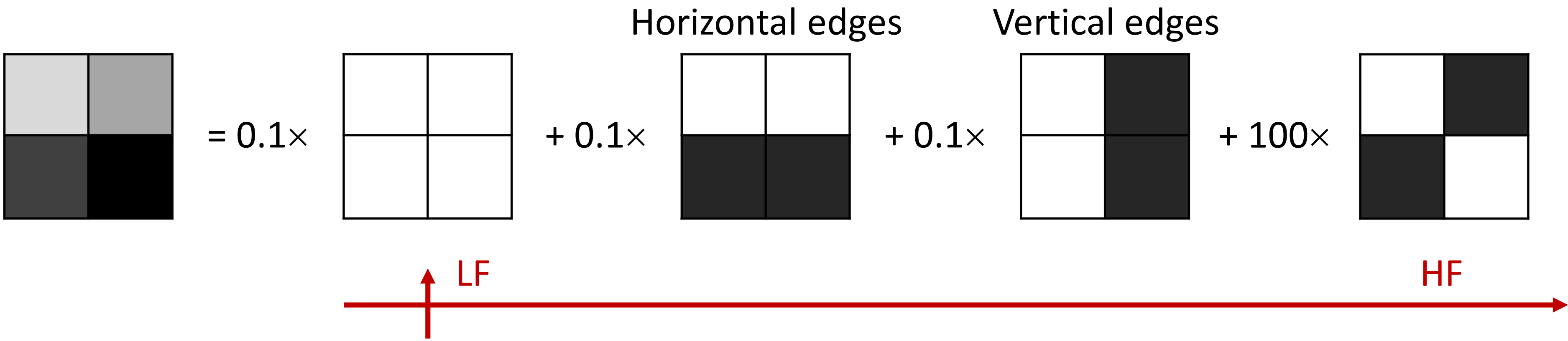
- Filtering process



- Will the output be very different from the input image after:
  - a HPF ?
  - A LPF ?

Why do we need transforms ?

- Filtering process

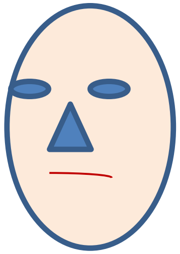


- Allows to have better understanding of the filtering process

Why do we need transforms ?

- Eigen-faces …

Eigen-faces



- Any human face can be regarded as a combination of some “standard faces” → eigen-faces.
  - For example, someone face might be composed of the “average face” plus 5% from eigenface 1, 0 % from eigenface 2, and so on …
- Usually to achieve a fair approximation of most faces not many eigenfaces are required.

Eigen-faces

 = a×  + b×  + c×  + d× $\square$

+ e× $\square$ + f× $\square$ + g× $\square$ + ...

Orthogonal Basis
Orthonormal Basis

Eigen-faces

- Orthogonal Basis
  - Orthogonality of two vectors: <***v***, ***w***>=0
  - {e$_k$}: $\langle \mathbf{e}_j, \mathbf{e}_k \rangle = \begin{cases} q(\mathbf{e}_k) & j = k \\ 0 & j \neq k \end{cases}$ where q(v) = |v|^2

- Orthonormal Basis
  - {e$_k$}: $\langle e_j, e_k \rangle = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}$
  - How to do the normalization?
    - $x_{norm} = \frac{x}{|x|} = (\frac{x_1}{|x|}, \frac{x_2}{|x|}, \cdots, \frac{x_n}{|x|})$

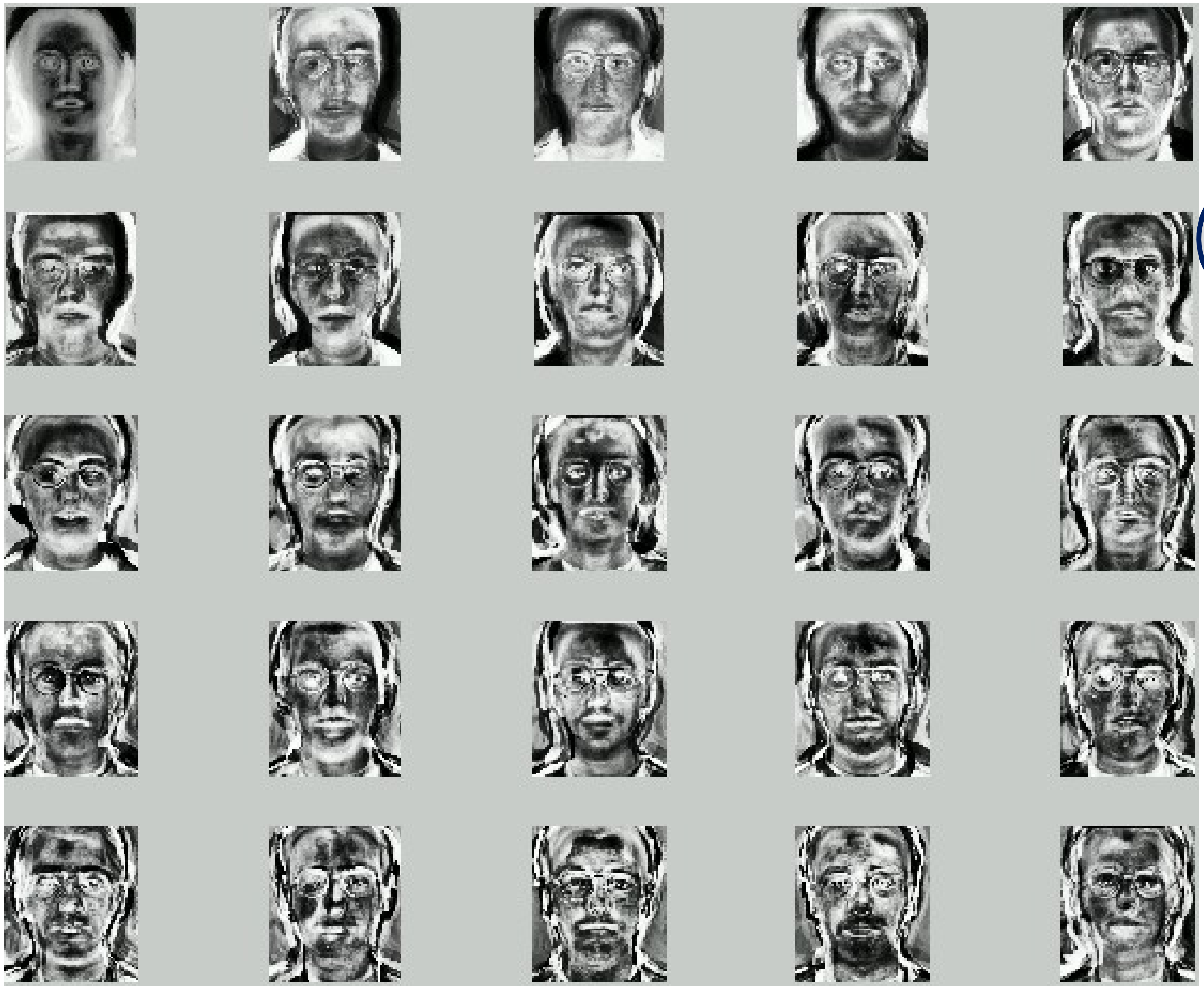
How to generate Eigen-faces ?

- A set of eigenfaces can be generated by performing a mathematical process called Principal Component Analysis (PCA) on a large set of images representing different faces.

- https://en.wikipedia.org/wiki/Eigenface

1. Prepare a training set of face images. The pictures constituting the training se
images. They must also be all resampled to a common pixel resolution ($r \times c$).
elements. For this implementation, it is assumed that all images of the training
2. Subtract the mean. The average image **a** has to be calculated and then subtra
3. Calculate the eigenvectors and eigenvalues of the covariance matrix **S**. Each ei
eigenvectors of this covariance matrix are therefore called eigenfaces. They ar
the practical applicability of eigenfaces stems from the possibility to compute
4. Choose the principal components. Sort the eigenvalues in descending order a
variance. Total variance $v = n \cdot (\lambda_1 + \lambda_2 + \ldots + \lambda_n)$, n = number of data ima
5. k is the smallest number satisfies : $\frac{n(\lambda_1 + \lambda_2 + \ldots + \lambda_k)}{v} > \epsilon$

Eigen-faces



The basis here is not about features we are familiar with, such as nose, eyes, etc, it is about "faces" that are linearly independents

Face recognition and classification

- Face classification/detection can be achieved by comparing how faces are represented by the basis (i.e., the weight of the eigen faces).

Name	Lenna
Photo	
Eigen weights	a, b, c, …

- Using eignface representation allows to record a person's face using a list of values (one value for each eigenface) → less space is taken to represent each person's face.

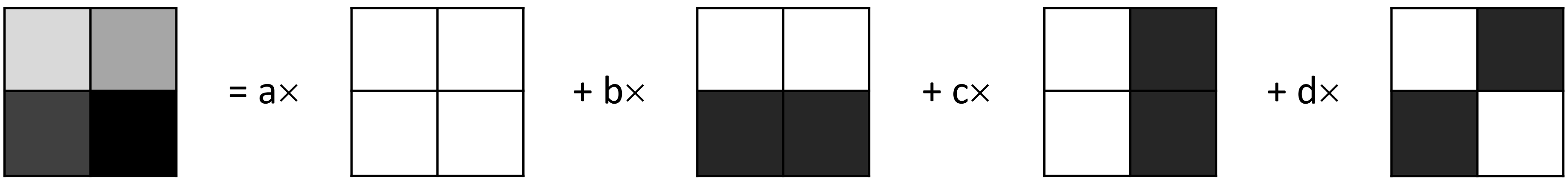
Eigen-faces

- How to get the eigen-faces (i.e., basis)?
  - Skipped
- How to get the weights of a face
  - Forward analysis
- How to generate a face with the given weights
  - Inverse analysis
- How to use Eigen-faces?
  - Classification
  - Compression

How do we build a basis for the image space

In general, how do we build a basis for the image space ?

$\vec{V} = a\hat{\imath} + b\hat{\jmath} + c\hat{k} + ..$



– A basis of images should have:
  - Linearly independent “images“ (avoid redundancy)
  - Enough elements in the basis to represent the whole image space → in principle, the number of images in the basis is equal to the number of pixels.

How to find the transformed domain version of an image

How to find the transformed domain version of an image

- Let us suppose we want to get b

$$\begin{array}{|c|c|}\hline 120 & 72 \\ \hline 40 & 24 \\ \hline\end{array} = a \times \begin{array}{|c|c|}\hline +1 & +1 \\ \hline +1 & +1 \\ \hline\end{array} + b \times \begin{array}{|c|c|}\hline +1 & +1 \\ \hline -1 & -1 \\ \hline\end{array} + c \times \begin{array}{|c|c|}\hline +1 & -1 \\ \hline +1 & -1 \\ \hline\end{array} + d \times \begin{array}{|c|c|}\hline +1 & -1 \\ \hline -1 & +1 \\ \hline\end{array}$$

How to find the transformed domain version of an image

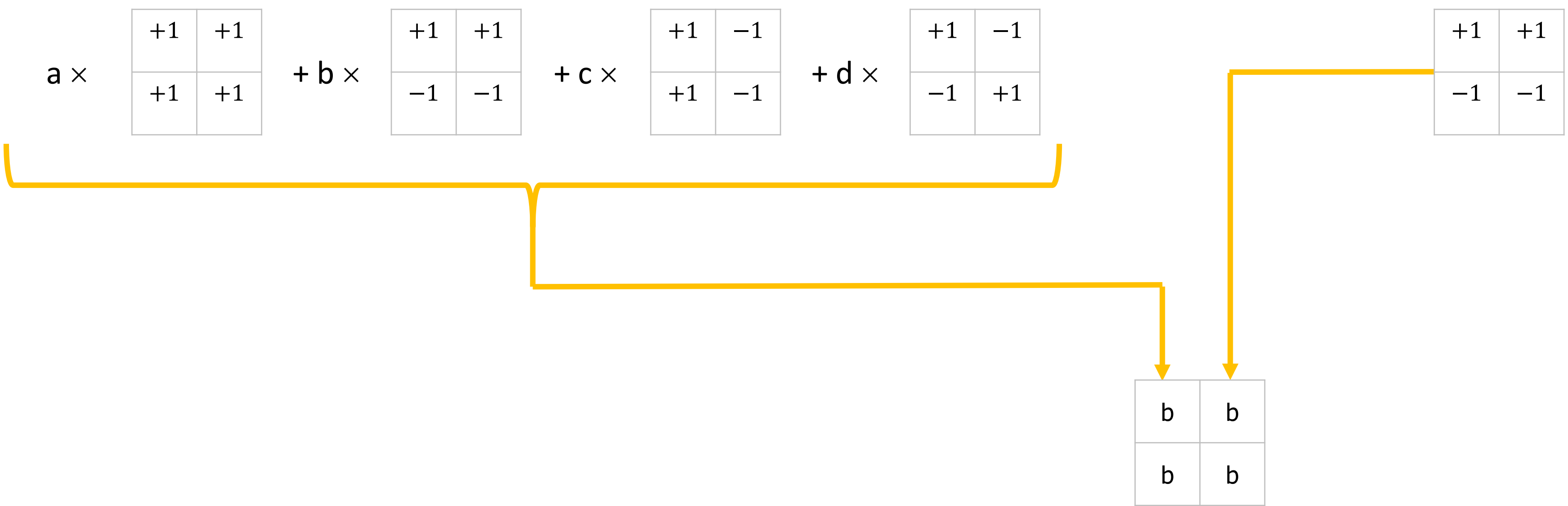
- Let us suppose we want to get b

$$\begin{array}{|c|c|}\hline 120 & 72 \\ \hline 40 & 24 \\ \hline \end{array} = a \times \begin{array}{|c|c|}\hline +1 & +1 \\ \hline +1 & +1 \\ \hline \end{array} + b \times \begin{array}{|c|c|}\hline +1 & +1 \\ \hline -1 & -1 \\ \hline \end{array} + c \times \begin{array}{|c|c|}\hline +1 & -1 \\ \hline +1 & -1 \\ \hline \end{array} + d \times \begin{array}{|c|c|}\hline +1 & -1 \\ \hline -1 & +1 \\ \hline \end{array}$$

$$\vec{V} = a\hat{\imath} + b\hat{\jmath} + c\hat{k} + ..$$

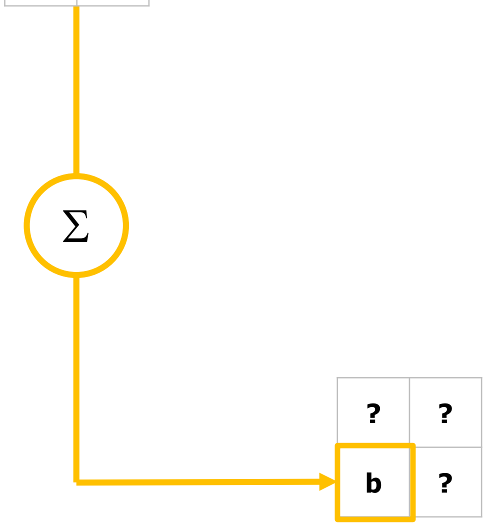
$$\boxed{\hat{\jmath}\vec{V} = b}$$

How to find the transformed domain version of an image



Concatenate to vector

So if we want *b* then we need to multiply the image with the corresponding basis , due the summation and divide by four



How to find the transformed domain version of an image

120	72
40	24

+1	+1
−1	−1

120	72
-40	-24

Element by element product

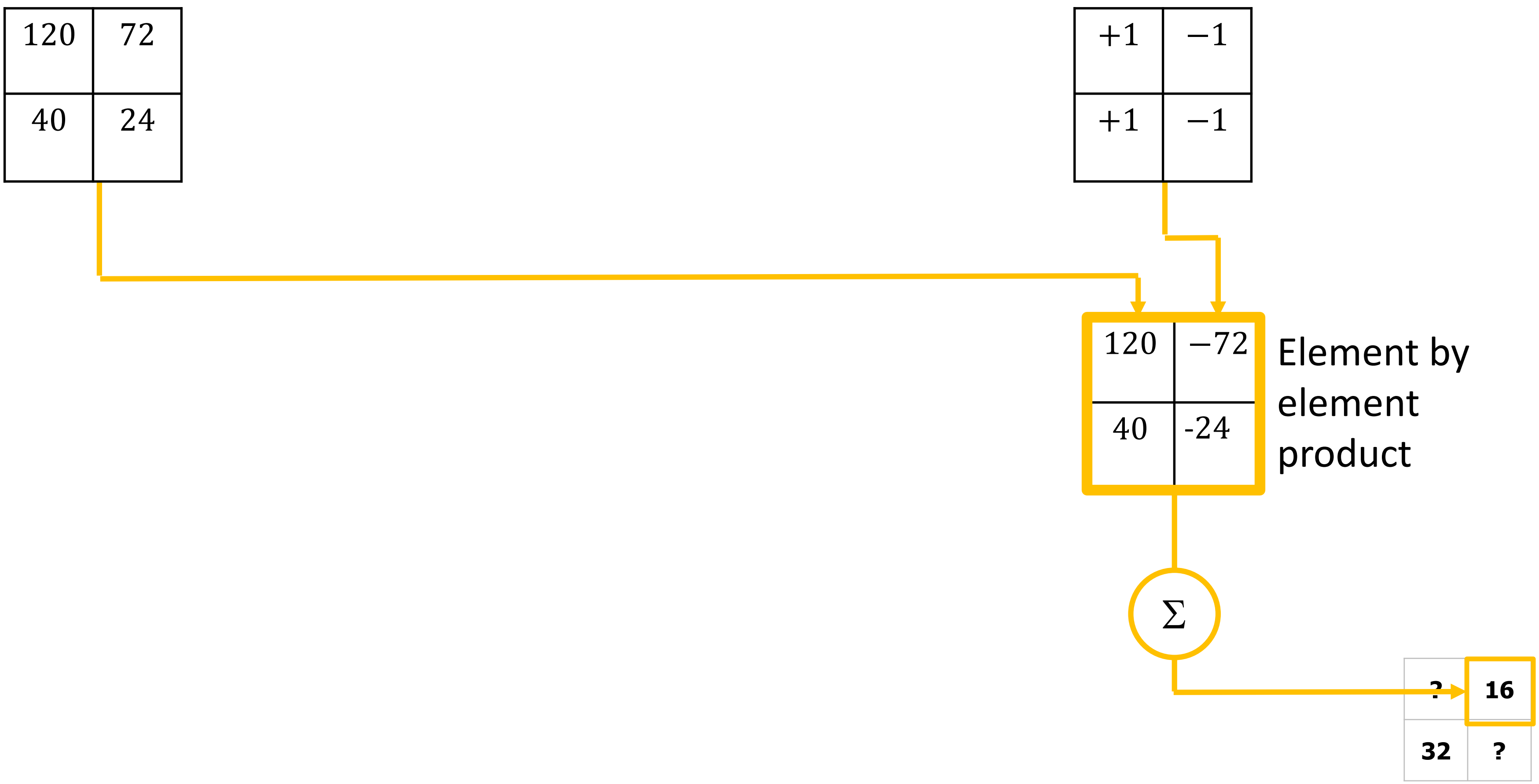
Σ

?	**?**
32	**?**

Concatenate to vector

So if we want *b* then we need to multiply the image with the corresponding basis , due the summation and divide by four

How to find the transformed domain version of an image



Discrete Fourier Transform

Discrete Fourier Transform

- Used in a wide range of applications: image analysis, filtering, reconstruction and compression
- Such transformations map a function to a set of coefficients of basis functions.
- The basis functions are sinusoidal and are therefore strongly localized in the frequency domain, but not in the time domain

1-D DFT

- Discrete Fourier Transform is used to decompose a signal into sine and cosine components

$$F(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j2\pi k \frac{n}{N}}$$

$$\boxed{e^{ix} = cosx + isinx}$$

$$= \sum_{n=0}^{N-1} x(n)W(n,k)$$

Complex value

Real

Imaginary

$$\boxed{x(n) = \sum_{k=0}^{N-1} F(k)e^{j2\pi n \frac{k}{N}} = \sum_{k=0}^{N-1} F(k)\left(\cos(2\pi n \frac{k}{N}) + j\sin(2\pi n \frac{k}{N})\right)}$$

1-D DFT in matrix notation

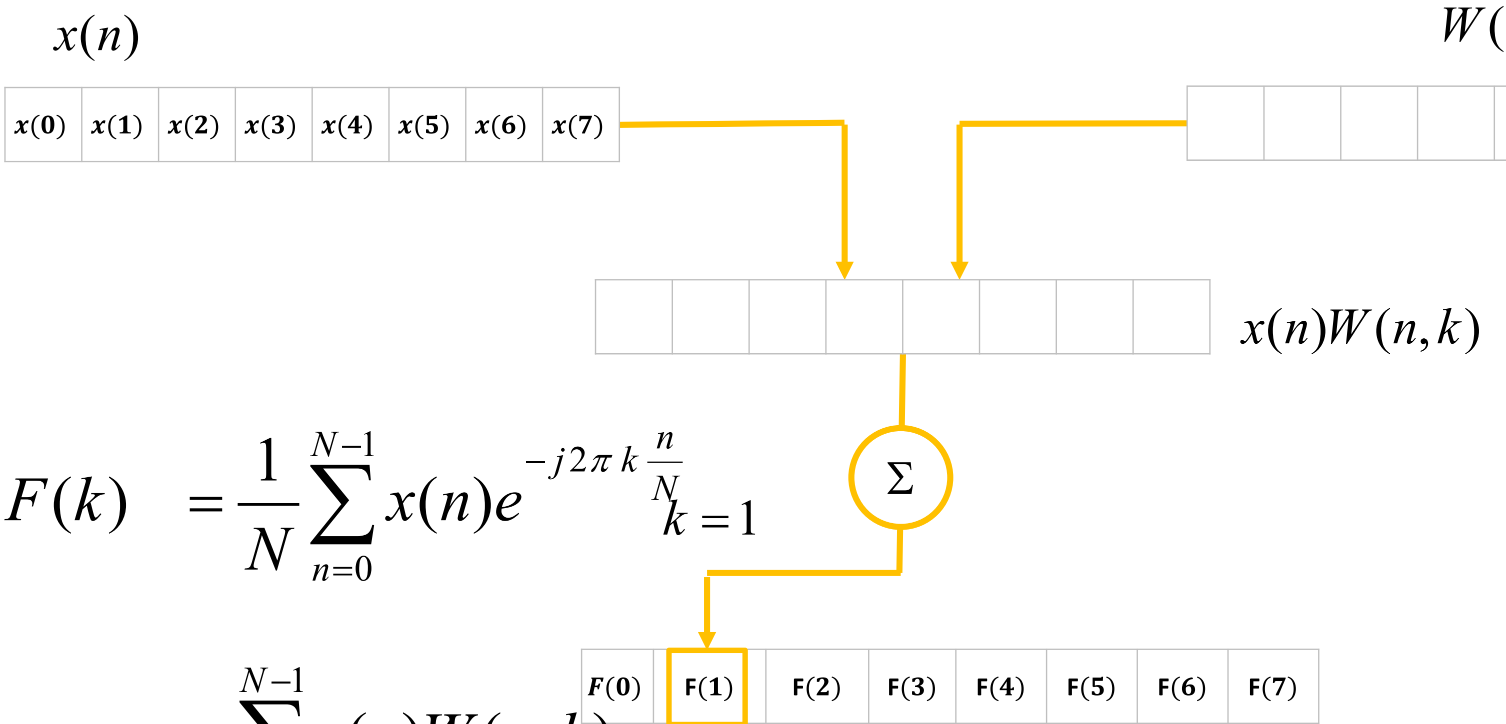
$$F(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}}$$

$$F(k) = \frac{1}{N}\begin{bmatrix} e^{-j2\pi k\frac{0}{N}} & e^{-j2\pi k\frac{1}{N}} & e^{-j2\pi k\frac{2}{N}} & \cdots & e^{-j2\pi k\frac{N-1}{N}} \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ \vdots \end{bmatrix} = \frac{1}{N} \overset{\mathbf{T}}{\boxed{\begin{bmatrix} e^{-j2\pi(0)\frac{0}{N}} & e^{-j2\pi(0)\frac{1}{N}} & e^{-j2\pi(0)\frac{2}{N}} & \cdots & e^{-j2\pi(0)\frac{N-1}{N}} \\ e^{-j2\pi(1)\frac{0}{N}} & e^{-j2\pi(1)\frac{1}{N}} & e^{-j2\pi(1)\frac{2}{N}} & \cdots & e^{-j2\pi(1)\frac{N-1}{N}} \\ e^{-j2\pi(2)\frac{0}{N}} & e^{-j2\pi(2)\frac{1}{N}} & e^{-j2\pi(2)\frac{2}{N}} & \cdots & e^{-j2\pi(2)\frac{N-1}{N}} \\ \vdots & \vdots & \vdots & & \vdots \end{bmatrix}}} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \end{bmatrix}$$

1-D DFT in matrix notation

$$W(n,k) = \left.\frac{e^{-j2\pi k \frac{n}{N}}}{N}\right|_{k=1}$$

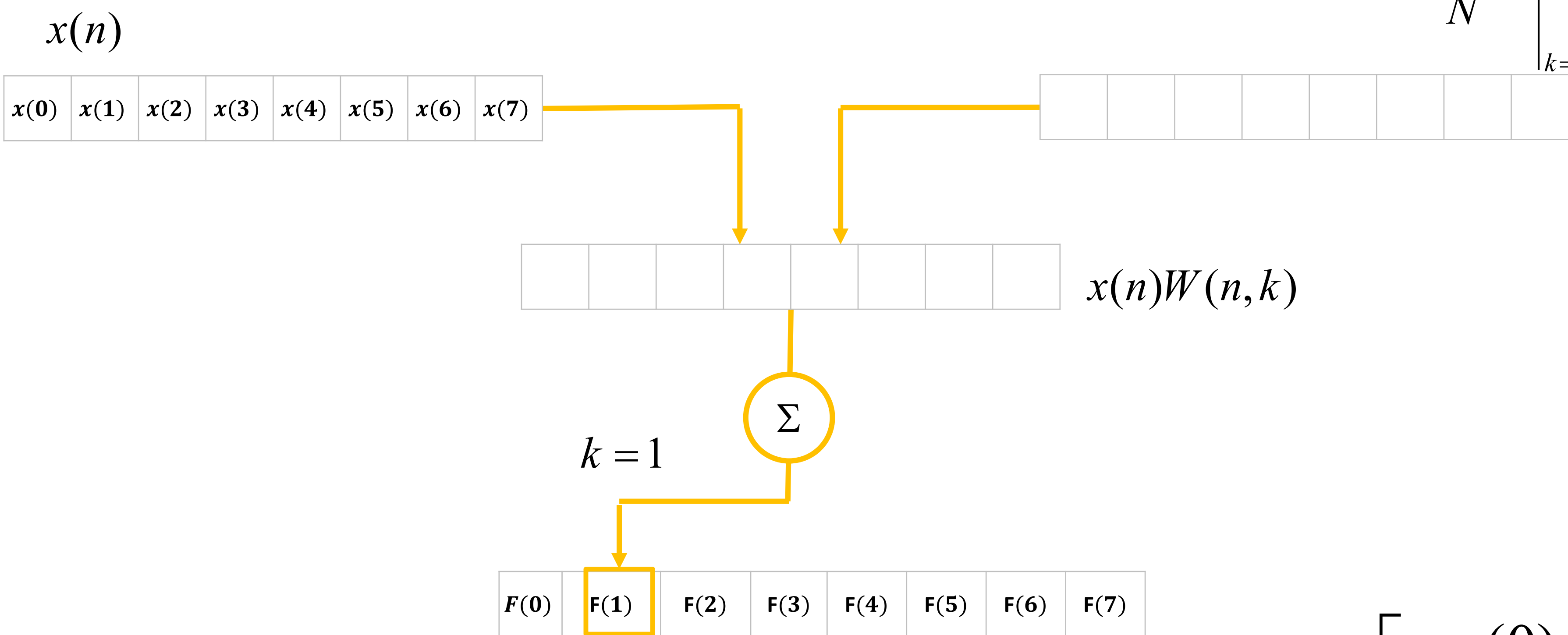


$$F(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}} \quad k = 1$$

$$= \sum_{n=0}^{N-1} x(n) W(n,k)$$

1-D DFT in matrix notation

$$W(n,k) = \left.\frac{e^{-j2\pi k \frac{n}{N}}}{N}\right|_{k=1}$$



$$F(k) = \frac{1}{N}\begin{bmatrix} e^{-j2\pi k\frac{0}{N}} & e^{-j2\pi k\frac{1}{N}} & e^{-j2\pi k\frac{2}{N}} & \dots & e^{-j2\pi k\frac{N-1}{N}} \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}$$

Orthogonality

Orthogonality

The vectors $u_k = \left[e^{\frac{2\pi i}{N} kn} \mid n = 0, 1, \ldots, N-1 \right]^T$ form an orthogonal basis over the set of N-dimensional complex vectors:

$$u_k^T u_{k'}^* = \sum_{n=0}^{N-1} \left(e^{\frac{2\pi i}{N} kn} \right) \left(e^{\frac{2\pi i}{N} (-k')n} \right) = \sum_{n=0}^{N-1} e^{\frac{2\pi i}{N} (k-k')n} = N \, \delta_{kk'}$$

where $\delta_{kk'}$ is the Kronecker delta. (In the last step, the summation is trivial if $k = k'$, where it is $1+1+\cdots=N$, and otherwise is a geometric series that can be explicitly summed to obtain zero.)

https://en.wikipedia.org/wiki/Discrete_Fourier_transform

Two-dimensional Discrete Fourier Transform (2D – DFT)

Two-dimensional DFT

- A rectangular image $x(n,m)$ of size $N \times N$ has the two-dimensional DFT (2-D DFT):

$$
\begin{aligned}
F(k,l) &= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(n,m) W(n,m,k,l) \\
&= \frac{1}{N^2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(n,m) e^{-j2\pi\left(\frac{kn}{N}+\frac{lm}{N}\right)}
\end{aligned}
$$

n row index, m column,
k row index, l column

Two-dimensional DFT

- The base functions are sine and cosine waves with increasing frequencies

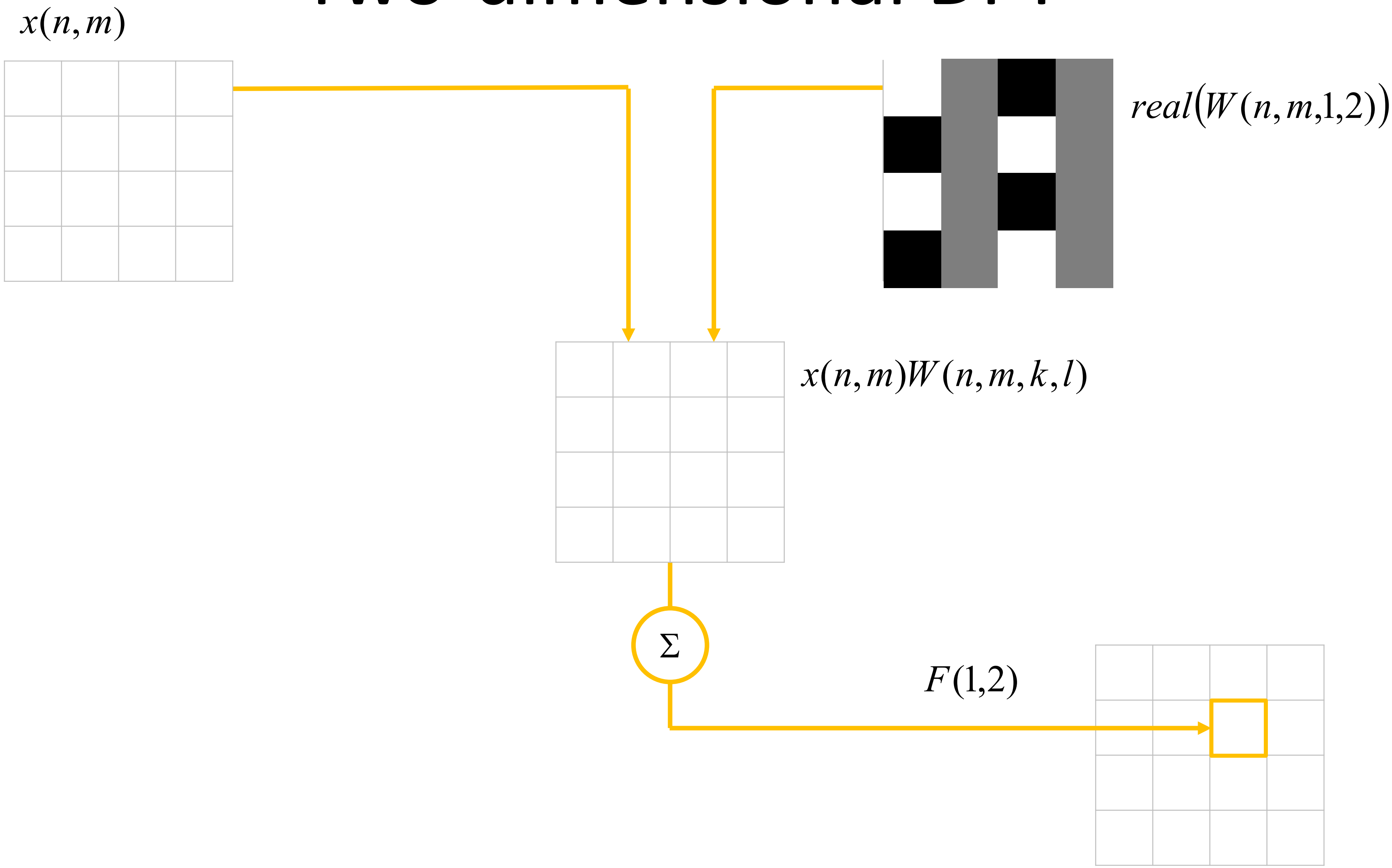
$$W(n,m,k,l)=\frac{1}{N^2}e^{-j2\pi\left(\frac{kn}{N}+\frac{lm}{N}\right)}$$

- *F(0,0)* represents the DC-component which corresponds to the average brightness

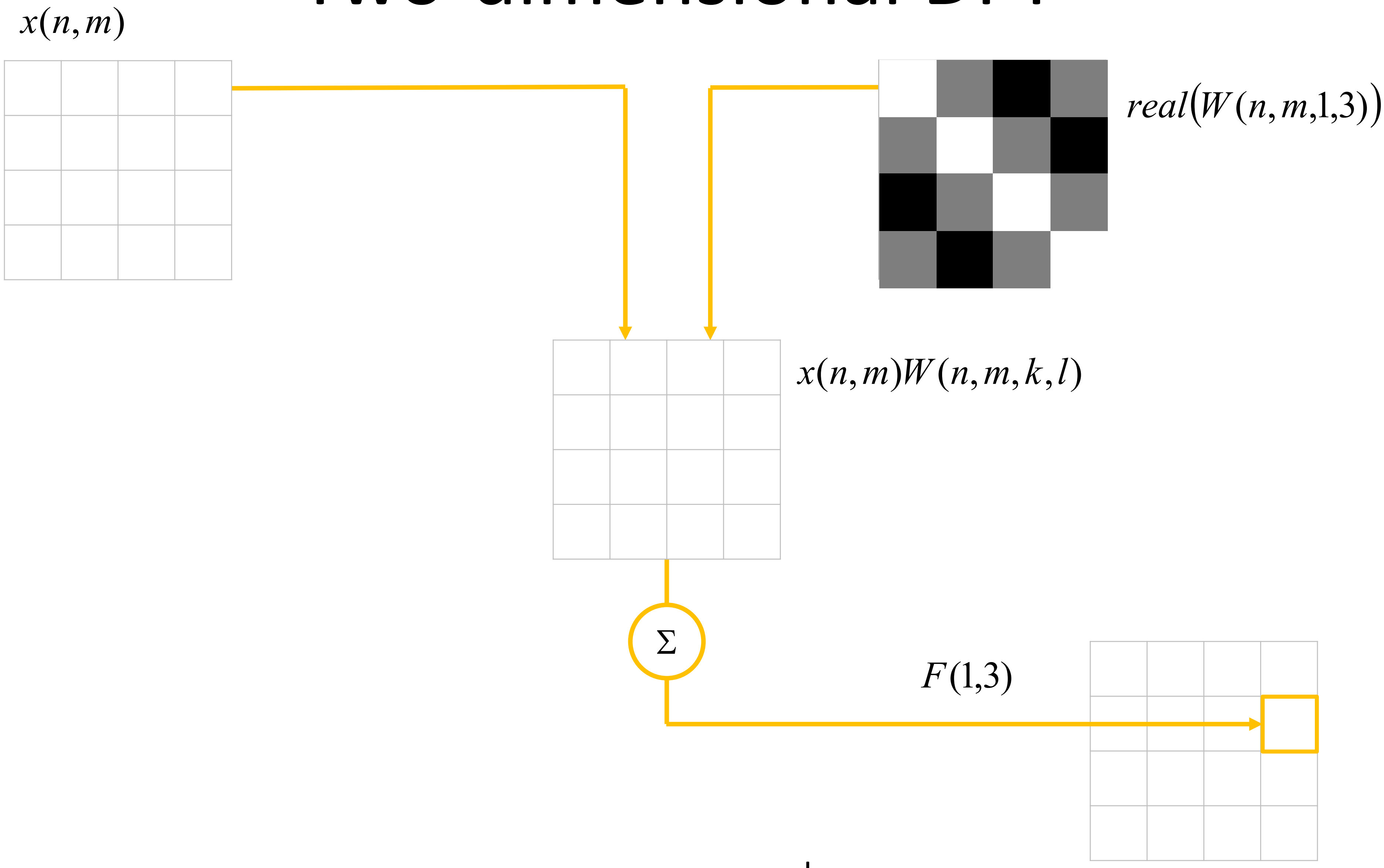
$$F(0,0)=\frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}x(n,m)$$

- *F(k,l)* (k>0, l>0)represents the AC-component

Two-dimensional DFT

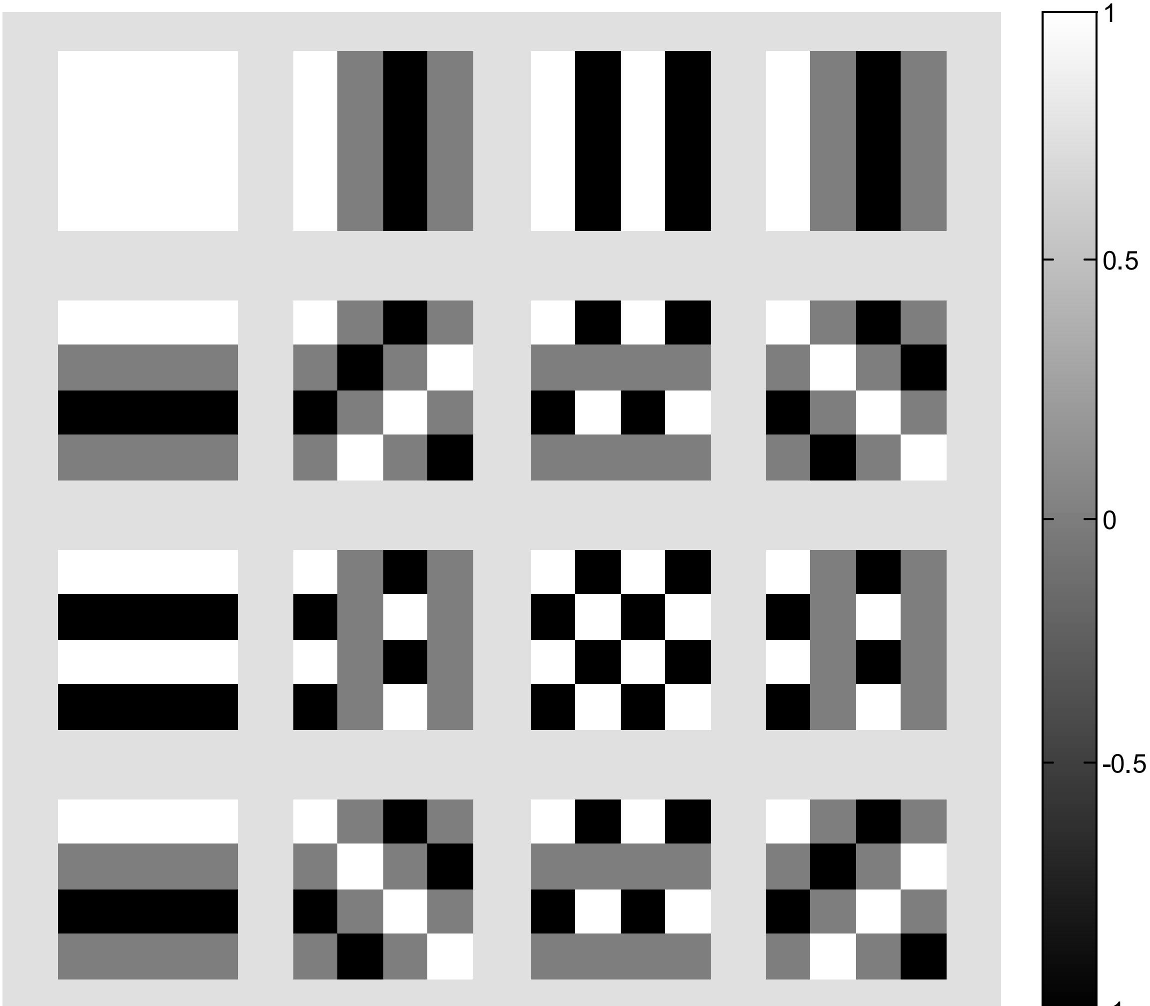


Two-dimensional DFT



code

Two-dimensional DFT



The left side images form a basis for the 4X4 image space

Basis functions of 4X4 2D-DFT ($real\{W(n,m,k,l)\}$)

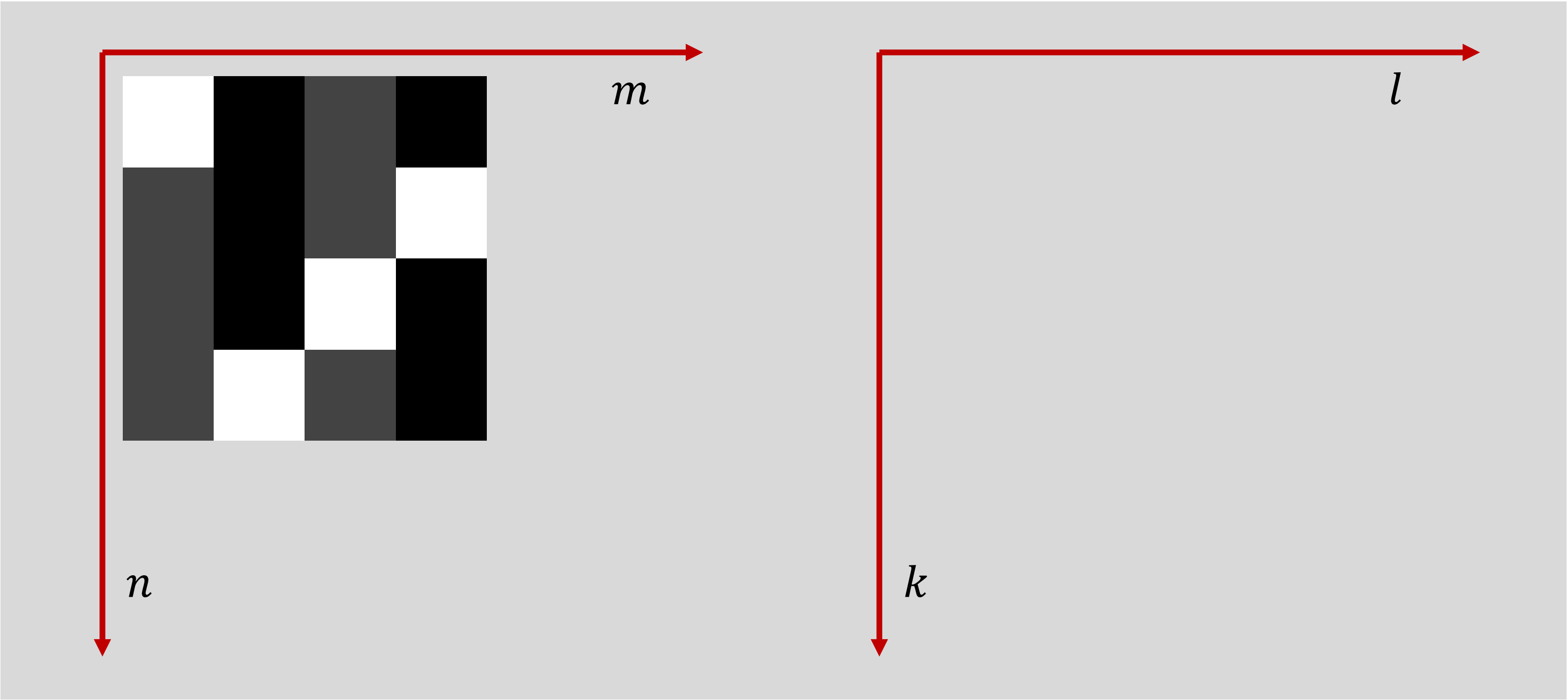
Any 4X4 image could be represented as a linear combination of the basis functions

When it is applied to an 4X4 image, it yields a 4X4 matrix of weighted values corresponding to how much of each basis function is present in the image

An 4x4 image that just contains one shade of gray will yield only a weighted value for the upper left hand DFT basis function (which has no frequencies in the x or y direction).

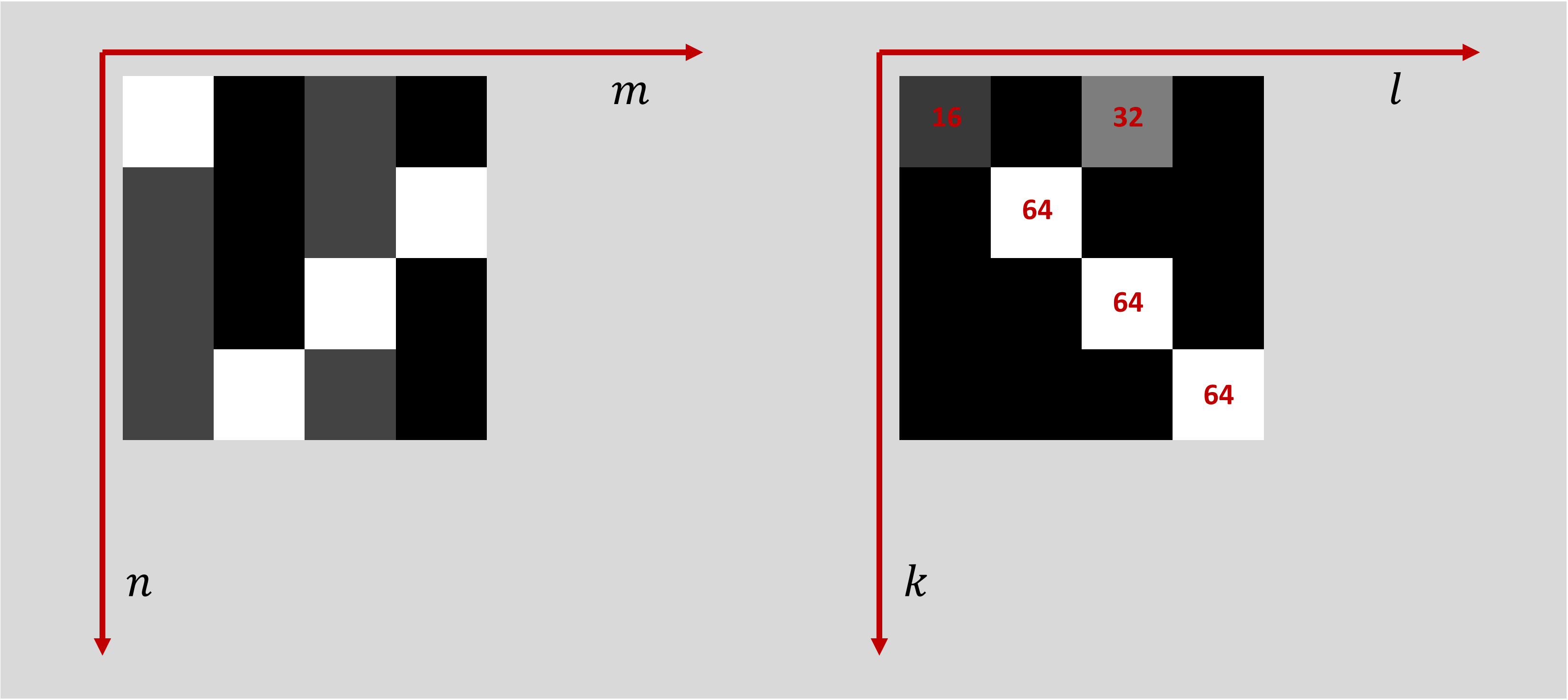
Two-dimensional DFT

- How the DFT of the following 4x4 image will look like ?

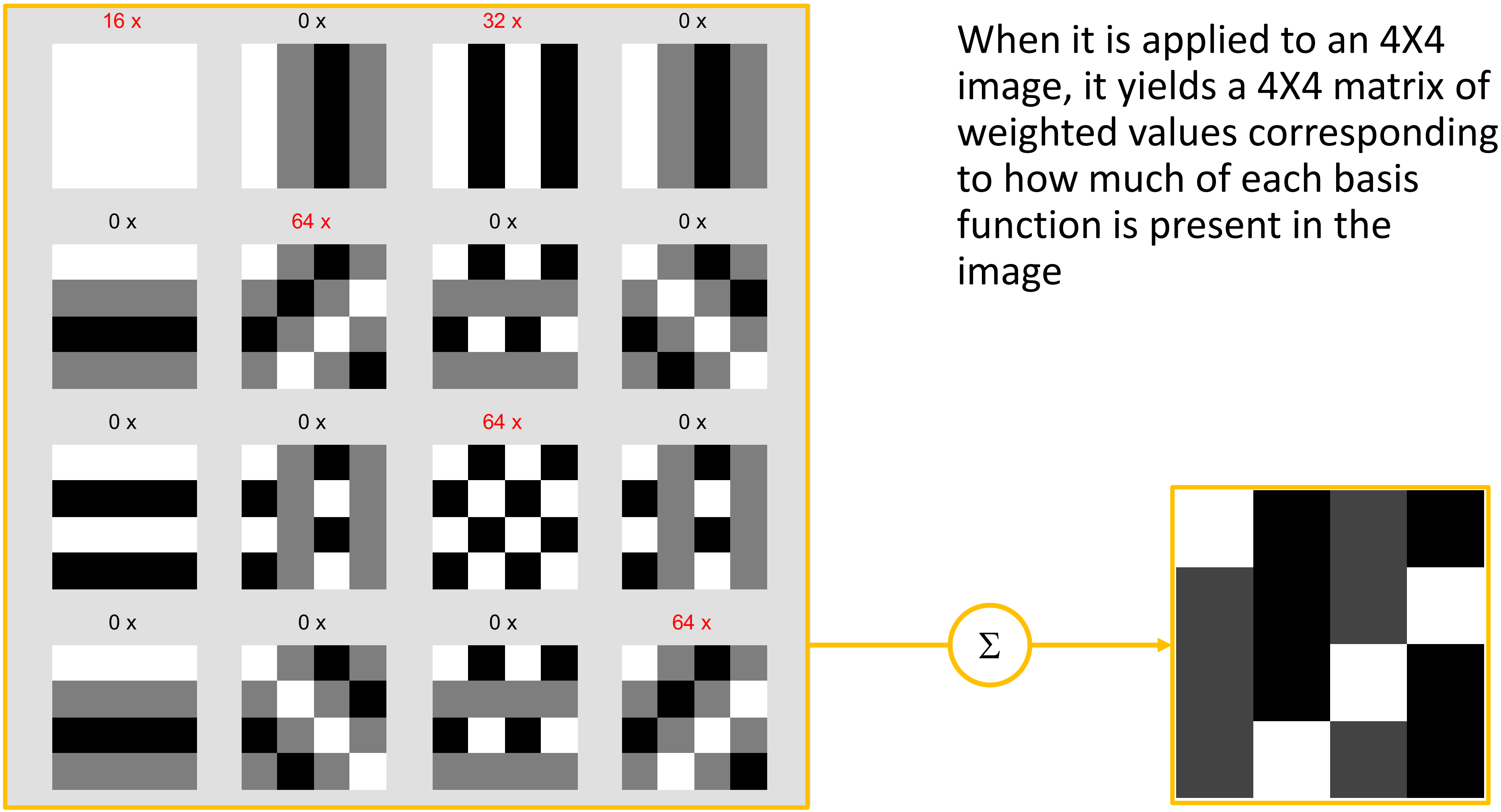


Two-dimensional DFT

- The forward (analysis) DFT of an 4x4 image
  - In this example the imaginary part is zero, thus it is not shown



Two-dimensional DFT



When it is applied to an 4X4 image, it yields a 4X4 matrix of weighted values corresponding to how much of each basis function is present in the image

The inverse (synthesis) DFT, which generates an image

Two-dimensional DFT

- The complexity of directly computing the 2D-DFT :
  - Each output point would require N^2 floating point multiplications
  - Since there are N^2 output points in a $N \times N$ image, the computational complexity of the DFT is $O(N^4)$
    - $N = 1024 \approx 1000 \rightarrow N^4 = 10^{12}$
  - FFT

Separability of 2-D DFT

- A double sum has to be calculated for each output-image point.
- However ...

$$
\begin{aligned}
F(k,l) &= \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m)e^{-j2\pi\left(\frac{kn}{N}+\frac{lm}{N}\right)} \\
&= \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m)e^{-j2\pi\left(\frac{kn}{N}\right)}e^{-j2\pi\left(\frac{lm}{N}\right)} \\
&= \frac{1}{N^2}\sum_{n=0}^{N-1}\left\{\left[\sum_{m=0}^{N-1} x(n,m)e^{-j2\pi\left(\frac{lm}{N}\right)}\right]e^{-j2\pi\left(\frac{kn}{N}\right)}\right\}
\end{aligned}
$$

Separability of 2-D DFT

- However, because the DFT is *separable*, it can be written as

$$
\begin{aligned}
F(k,l) &= \frac{1}{N^2}\sum_{n=0}^{N-1}\left\{\left[\sum_{m=0}^{N-1} x(n,m)e^{-j2\pi\left(\frac{lm}{N}\right)}\right]e^{-j2\pi\left(\frac{kn}{N}\right)}\right\} \\
&= \frac{1}{N}\sum_{n=0}^{N-1}\left\{P(n,l)e^{-j2\pi\left(\frac{kn}{N}\right)}\right\} \\
P(n,l) &= \frac{1}{N}\sum_{m=0}^{N-1} x(n,m)e^{-j2\pi\left(\frac{lm}{N}\right)}
\end{aligned}
$$

Separability of 2-D DFT

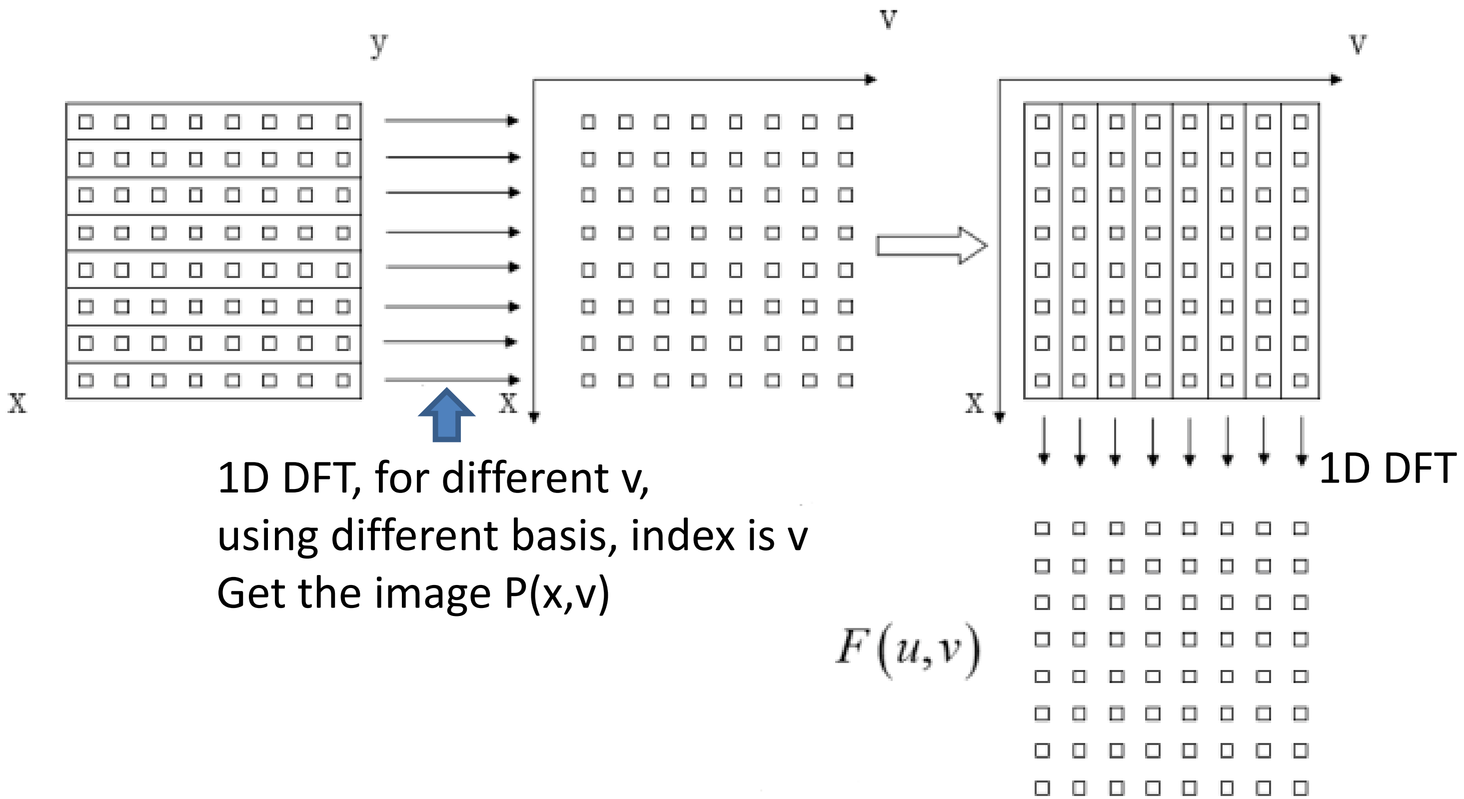
- The spatial domain image is firstly transformed into an intermediate image using 1-D DFT applied to the rows (columns)

$$P(n,l) = \frac{1}{N}\sum_{m=0}^{N-1} x(n,m) e^{-j2\pi\left(\frac{lm}{N}\right)}$$

- This intermediate image is then transformed into the final image, again using 1-D DFT applied to columns (rows)

$$F(k,l) = \frac{1}{N}\sum_{n=0}^{N-1} P(n,l) e^{-j2\pi\left(\frac{kn}{N}\right)}$$

Separability of 2-D DFT



1D DFT, for different v,
using different basis, index is v
Get the image P(x,v)

Properties of the 2D-DFT

- The DFT and its inverse are periodic:
  - Let us look at the 1D-DFT case:

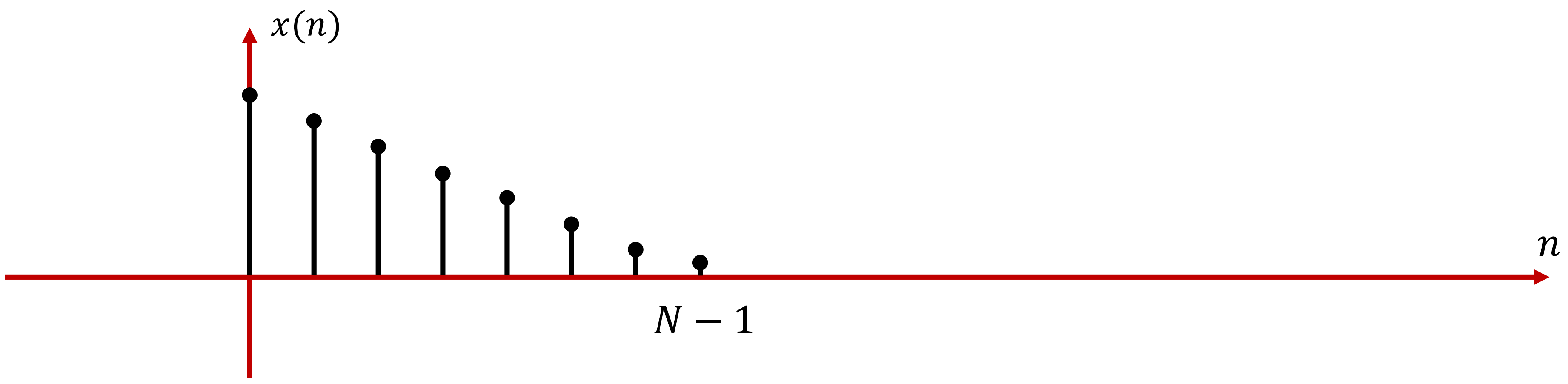
$$F(k+N) \quad = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi n \frac{k+N}{N}} \quad = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi n \frac{k}{N}} e^{-j2\pi n} \quad = F(k)$$

$$x(n+N) \quad = \sum_{k=0}^{N-1} F(k) e^{j2\pi k \frac{n+N}{N}} \quad = x(n)$$

  - The period is *N*

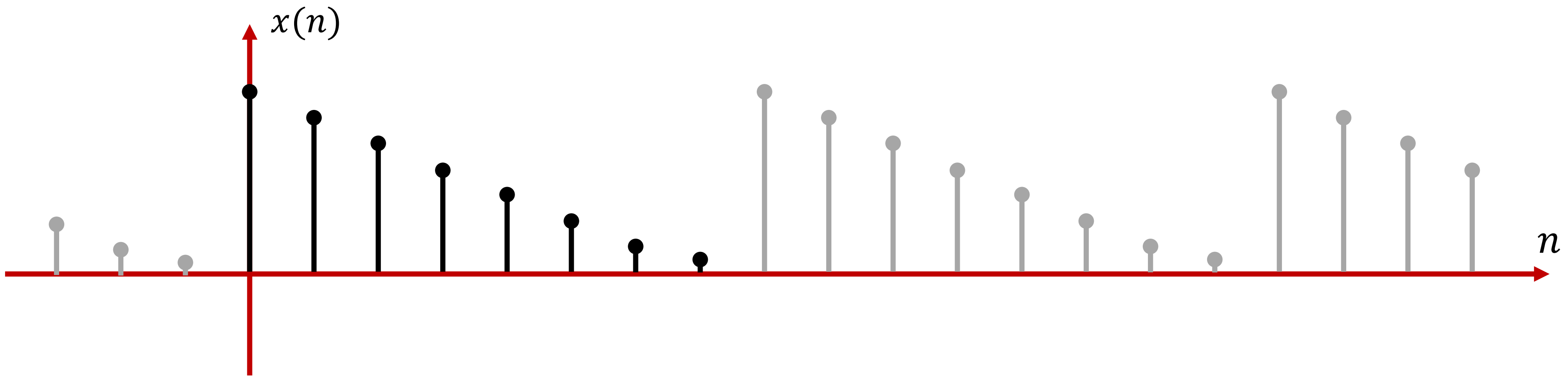
Properties of the 2D-DFT

- If x(n) is a finite domain function shown in the following :



Properties of the 2D-DFT

- The use of DFT to represent a finite domain function can be thought of as implicitly defining an extension of the function outside the domain.
- Once we write a function as a sum of complex exponential functions we can evaluate that sum at any point *n*, even where the original function x(n) was not declared.



Matlab : Computing 2D-DFT

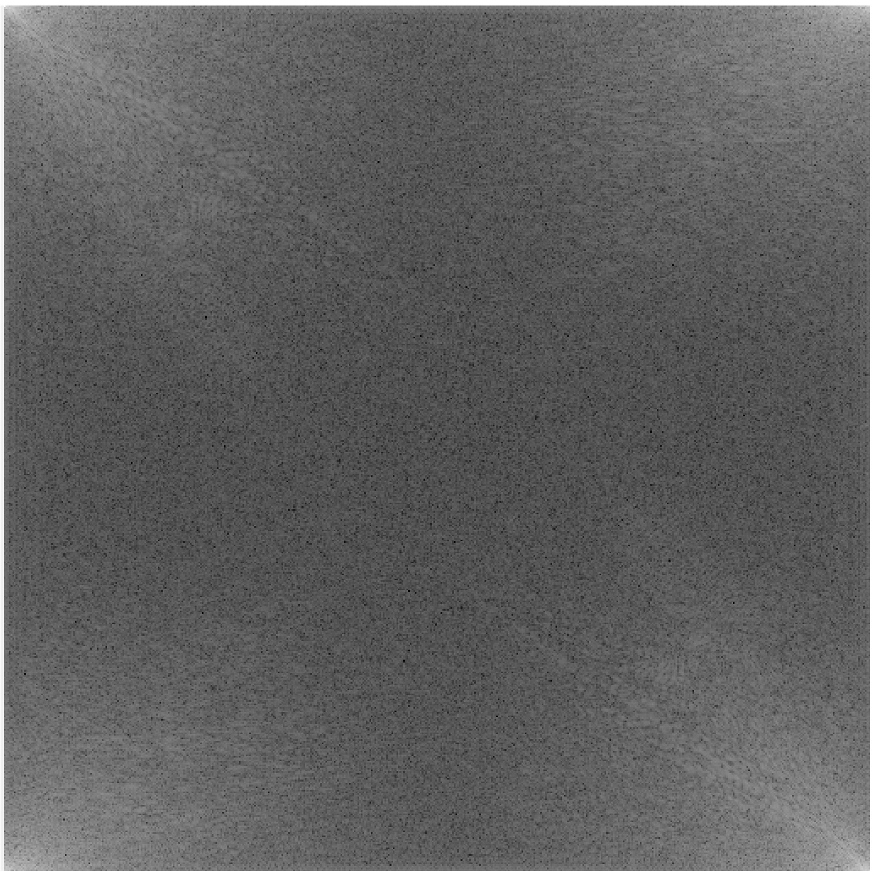
- 2D DFT and inverse Fourier transforms are implemented in fft2 and ifft2, respectively
- fftshift: Move origin (DC component) to image center for display
- Example:

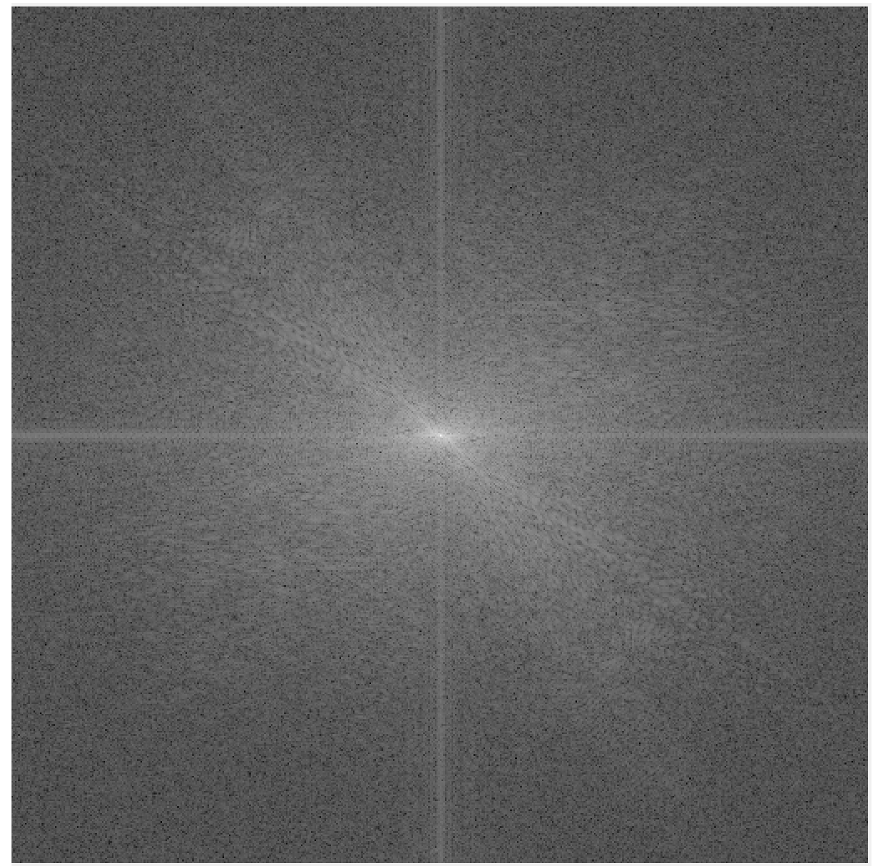
```
>> %I = imread('test.png'); generate an image
>> F = fftshift(fft2(I));
>> imshow(log(abs(F)),[]);
>> imshow(angle(F),[]);
```

`u = real(z), v = imag(z), r = abs(z),` and `theta = angle(z)`

Matlab : Computing 2D-DFT

- `fftshift`





Properties of 2-D DFT

- The DFT produces a complex valued matrix "image"
- It is displayed with two images, typically *magnitude* and *phase*, however, usually only the magnitude is displayed
- The Fourier domain image has a much greater range than the image in the spatial domain. Hence, its values are usually calculated and stored in float values and represented in log-scale

Properties of the 2D-DFT

- The DFT / 2D-DFT produces complex values
- It is displayed with two images, typically *magnitude* and *phase.*

$$F(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}} = \left|F(k)\right| e^{j\angle F(k)}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} x(n)\left(\cos(2\pi k \frac{n}{N}) - j\sin(2\pi k \frac{n}{N})\right)$$

Properties of the 2D-DFT

- For real images, the DFT is symmetrical about the origin (or N/2)

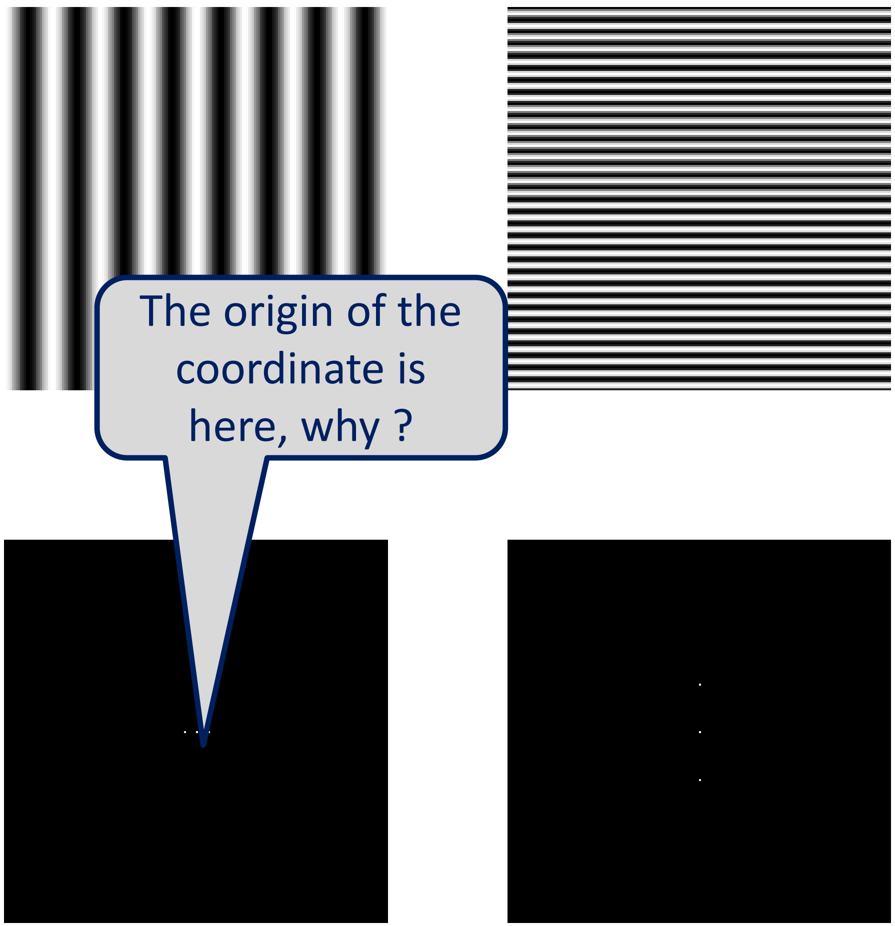
$$F(k,l) = \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m) e^{-j2\pi\left(\frac{kn}{N}+\frac{lm}{N}\right)}$$

$$F(N-k,N-l) = \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m) e^{-j2\pi\left(\frac{(N-k)n}{N}+\frac{(N-l)m}{N}\right)}$$

$$= \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m) e^{-j2\pi\left(\frac{(-k)n}{N}+\frac{(-l)m}{N}\right)}$$

$$= \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m) e^{+j2\pi\left(\frac{kn}{N}+\frac{lm}{N}\right)} = F^*(k,l)$$

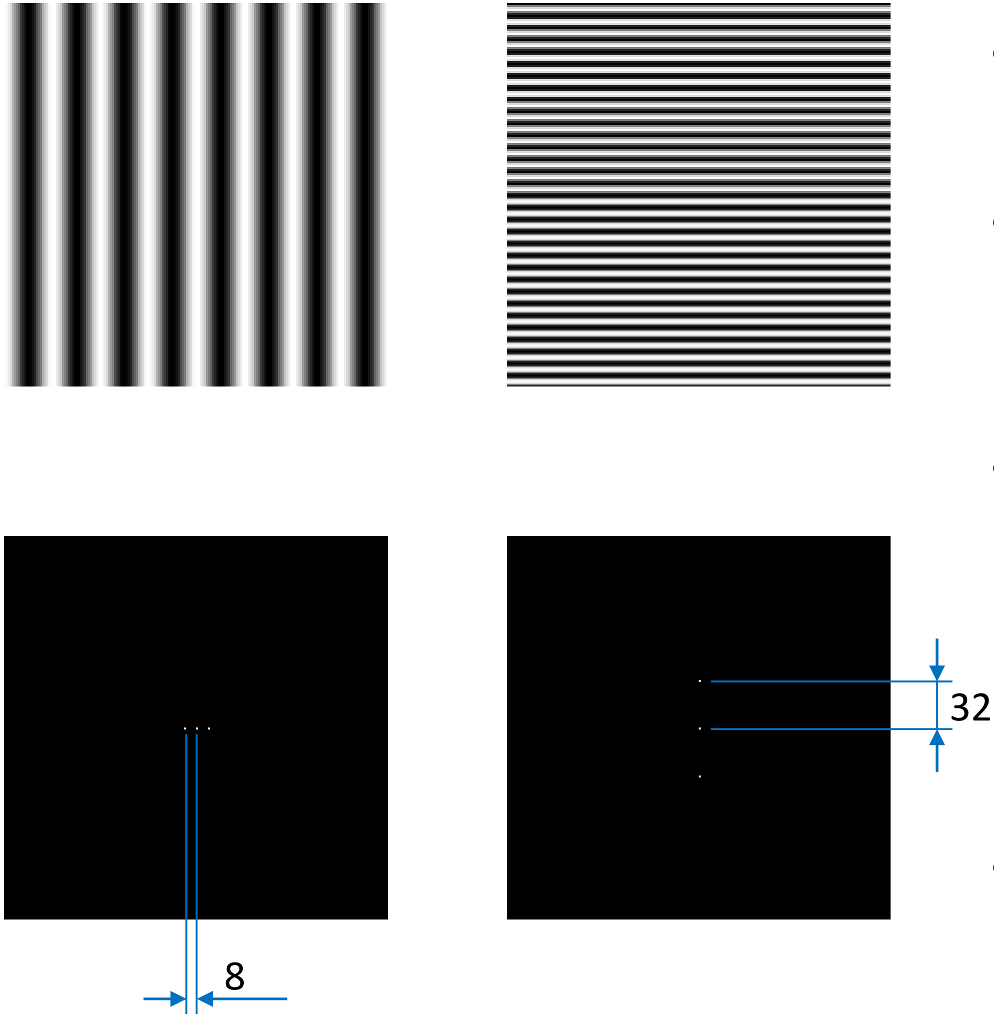
2-D DFT : example 1



code

- Images that are pure cosines have particularly simple DFT
- Pure horizontal cosine of 8 cycles and pure vertical cosine of 32 cycles.
- The DFT just has a single component, represented by 2 bright spots symmetrically placed about the center of the FT image
- The center of the image is the origin of the frequency coordinate system.

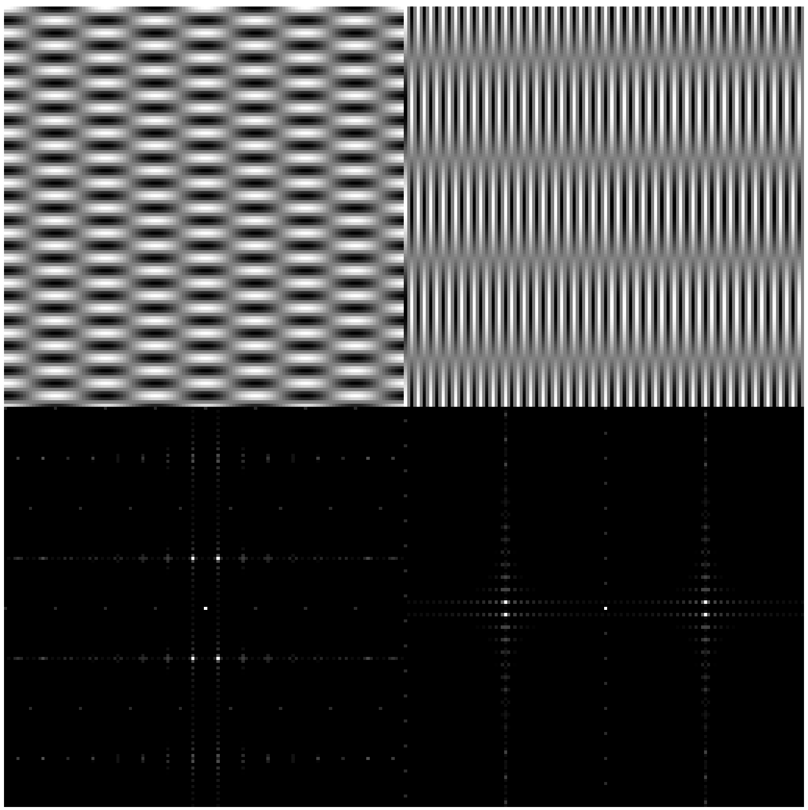
2-D DFT : example 1



code

- Images that are pure cosines have particularly simple DFT
- Pure horizontal cosine of 8 cycles and pure vertical cosine of 32 cycles.
- The DFT just has a single component, represented by 2 bright spots symmetrically placed about the center of the FT image
- The center of the image is the origin of the frequency coordinate system.

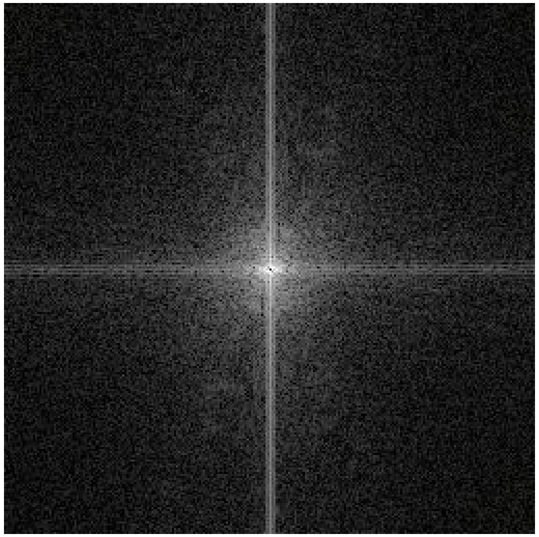
2-D DFT : example 1



- Images of 2D cosines with both horizontal and vertical components.
- (left) 4 cycles horizontal and 16 cycles vertically.
- (right) 32 cycles horizontally and 2 cycles vertically
- For real images, the FT is symmetrical about the origin so the 1st and 3rd (2nd and 4th) quadrants are the same

2-D DFT : example 2

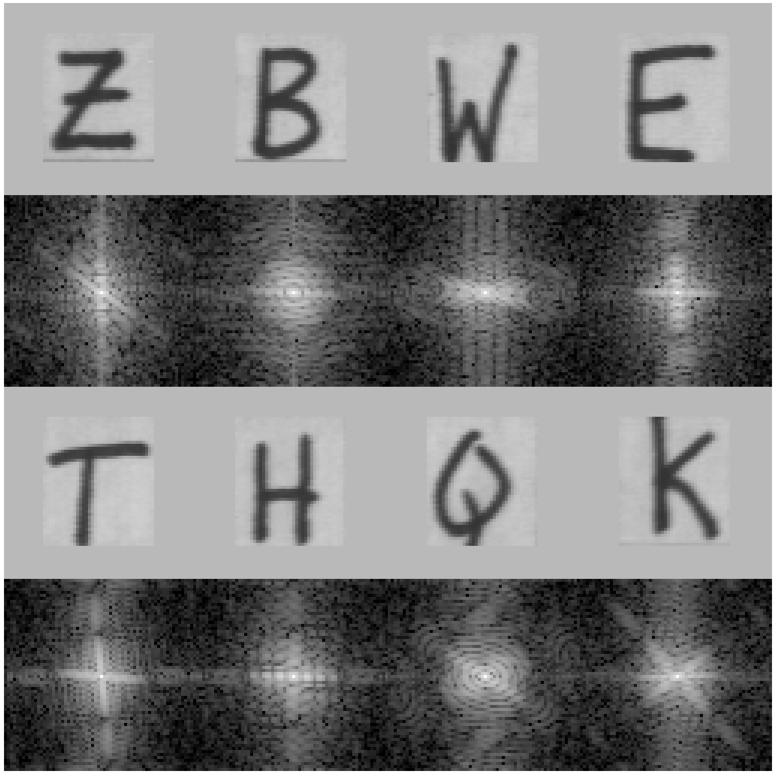
(a) Chest radiograph



(b) 2-D Fourier spectrum of (a)

broad range of spatial frequencies
significant vertical and horizontal features, due to ribs and vertebral column

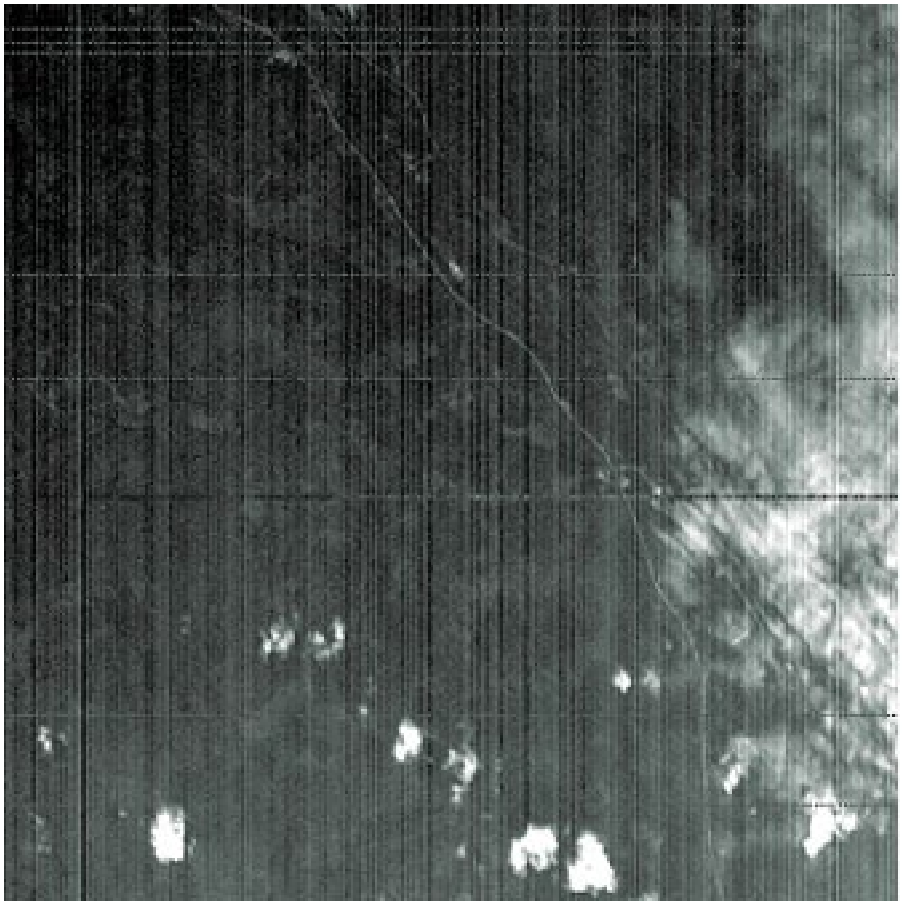
2-D DFT: example 3



The DFTs tend to have bright lines perpendicular to lines in the original letter.

If the letter has circular segments, then so does the FT.

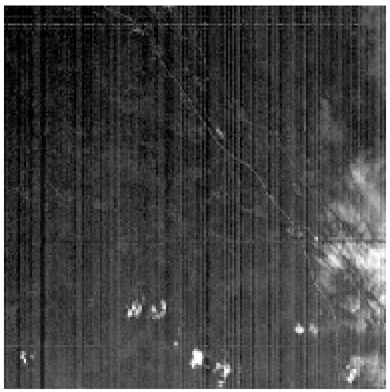
Noise reduction using DFT

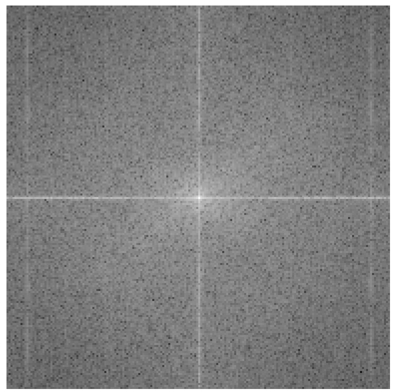


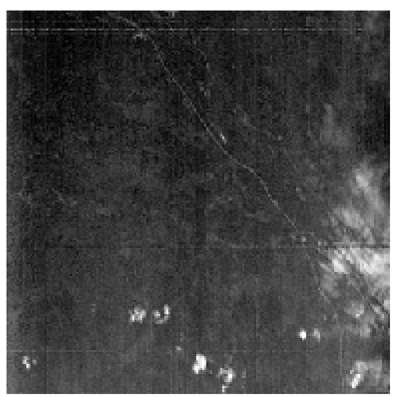
code

- Hyperspectral images can present semi-periodic partially deterministic disturbance patterns, which come from the image formation process and are characterized by a high degree of spatial and spectral coherence.
- **Vertical striping**: problem of CHRIS images; changes in temperature produce a dilation of the slit that results in a complex vertical pattern dependent on the sensor's temperature this hamper the operational use of CHRIS images since latter processing stages are drastically affected by these anomalous pixels.
- These errors must be identified and corrected by making use of both spatial and spectral information of the anomalous pixel and its neighbors.

Noise reduction using DFT

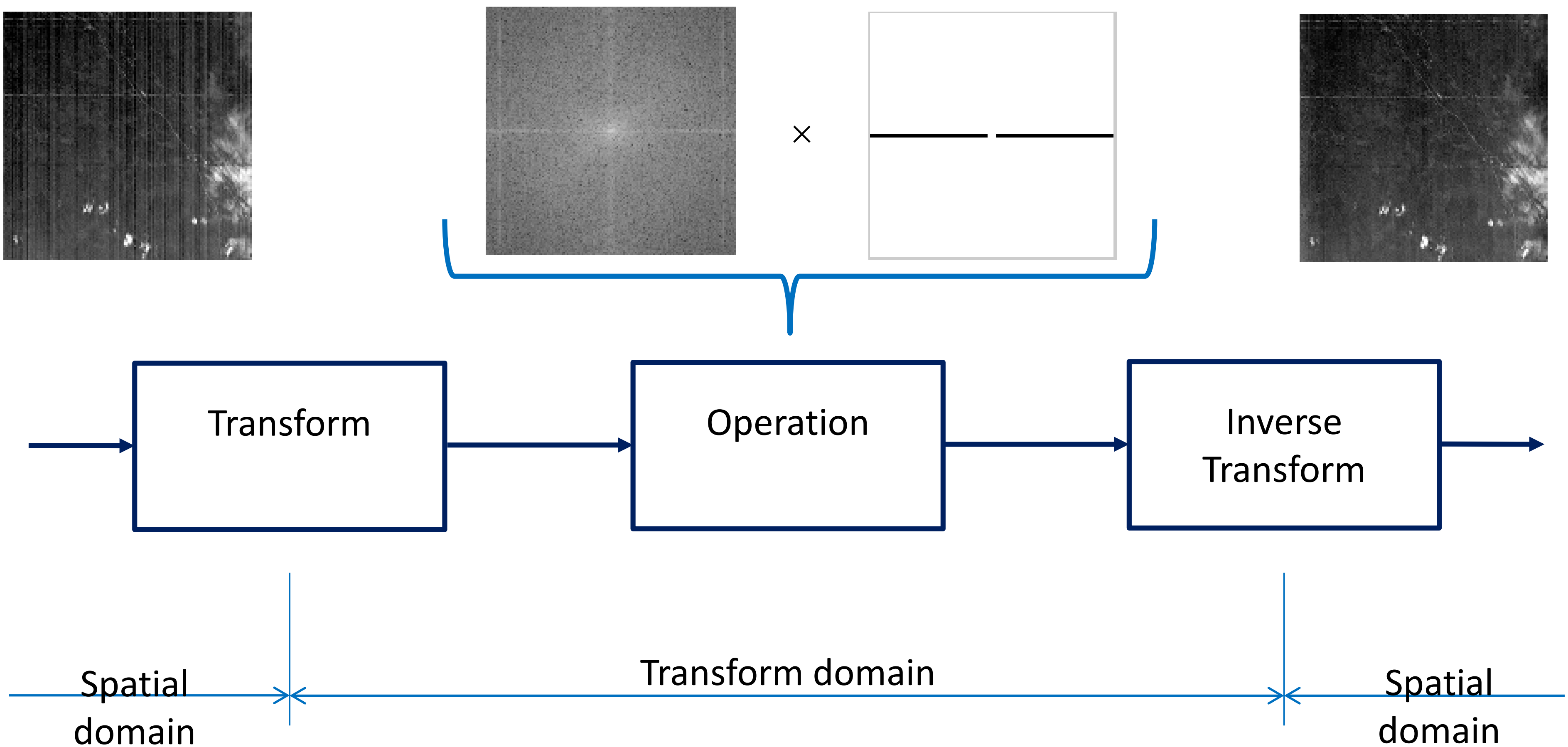






- Hyperspectral images can present semi-periodic partially deterministic disturbance patterns, which come from the image formation process and are characterized by a high degree of spatial and spectral coherence.
- Vertical striping: problem of CHRIS images; changes in temperature produce a dilation of the slit that results in a complex vertical pattern dependent on the sensor's temperature this hamper the operational use of CHRIS images since latter processing stages are drastically affected by these anomalous pixels.
- These errors must be identified and corrected by making use of both spatial and spectral information of the anomalous pixel and its neighbors.

Noise reduction using DFT



Some limitations of DFT for image processing

- To evaluate the DFT we need to work with complex numbers → due to the multiplication with the complex kernel it is more difficult for implementation
- Both *magnitude* and phase information are relevant for the reconstruction of the image.
- So when processing and image in the DFT domain we need to understand what will happen to both the *magnitude* and phase
  - Difficult to handle and display

Why do we have the imaginary part in the DFT ?

- Let us look at the 1D-DFT case:

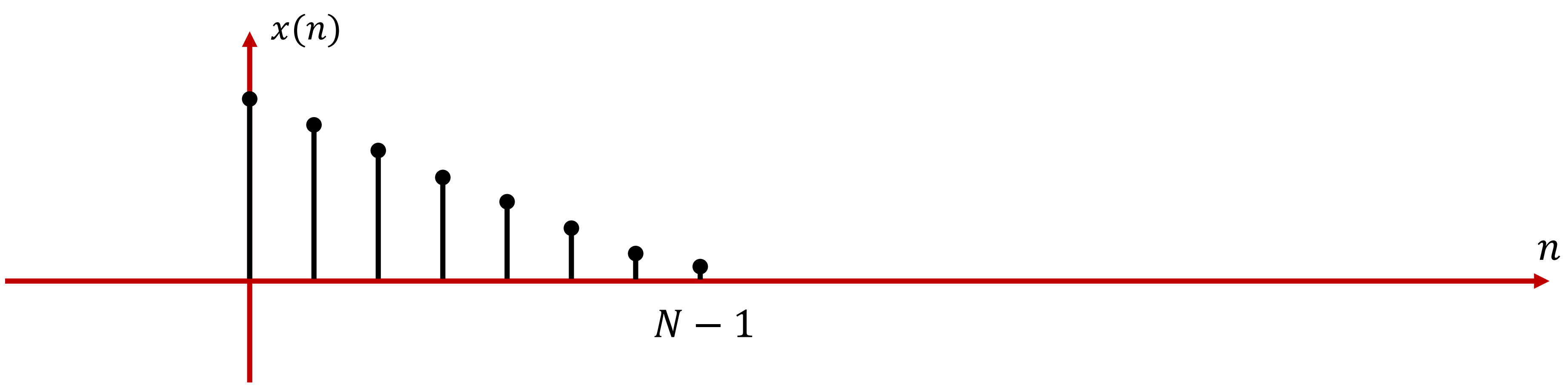
$$F(k)=\frac{1}{N}\sum_{n=0}^{N-1}x(n)e^{-j2\pi k\frac{n}{N}}=\frac{1}{N}\sum_{n=0}^{N-1}x(n)\left(\cos(2\pi k\frac{n}{N})-j\sin(2\pi k\frac{n}{N})\right)$$

- Cosine waves are due to the even component of the signal $x(n)$
- Sine waves are due to the odd component of the signal $x(n)$

How to get rid of the imaginary part in the DFT ?

How to get rid of the imaginary part in the DFT ?

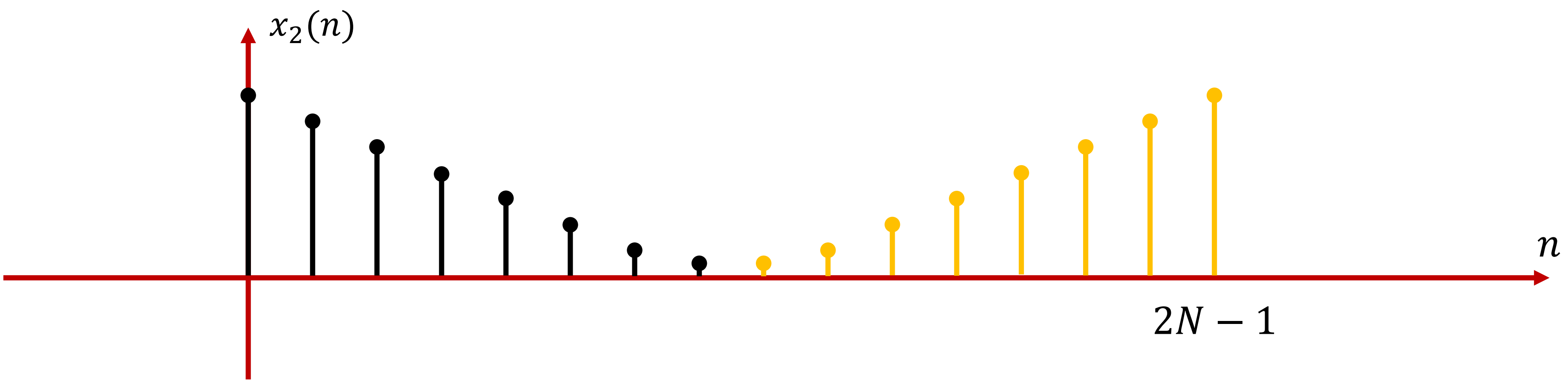
- We need to have an even symmetry signal
- How to make the following signal even symmetry signal ?



How to get rid of the imaginary part in the DFT ?

- Let suppose that a new signal is generated from $x(n)$ such that:

$$x_2(n) = \begin{cases} x(n) & ; \quad 0 \le n \le N-1 \\ x(2N-1-n) & ; \quad N \le n \le 2N-1 \end{cases}$$



How to get rid of the imaginary part in the DFT ?

$$F(k) = \frac{1}{2N}\sum_{n=0}^{2N-1} x_2(n)e^{-j2\pi k\frac{n}{2N}} = \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)e^{-j2\pi k\frac{n}{2N}} + \sum_{n=N}^{2N-1} x(2N-1-n)e^{-j2\pi k\frac{n}{2N}}\right)$$

$$= \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)\left(e^{-j2\pi k\frac{n}{2N}} + e^{-j2\pi k\frac{(2N-1-n)}{2N}}\right)\right)$$

① → $$= \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)\left(e^{-j2\pi k\frac{n}{2N}} + e^{j2\pi k\frac{(1+n)}{2N}}\right)\right)$$

② → $$= \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)e^{j\pi k\left(\frac{1}{2N}\right)}\left(e^{-j\pi k(\frac{2n+1}{2N})} + e^{j\pi k(\frac{2n+1}{2N})}\right)\right)$$

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

① → ② :

$$e^{-j2\pi k\frac{n}{2N}} + e^{j2\pi k\frac{1+n}{2N}}$$

$$= e^{j2\pi k\frac{1}{4N}}\left(e^{-j2\pi k\frac{2n+1}{4N}} + e^{j2\pi k\frac{2n+1}{4N}}\right)$$

$$= e^{j\pi k\frac{1}{2N}}\underbrace{\left(e^{-j\pi k\frac{2n+1}{2N}} + e^{j\pi k\frac{2n+1}{2N}}\right)}_{\cos}$$

How to get rid of the imaginary part in the DFT ?

$$
\begin{aligned}
F(k) &= \frac{1}{2N}\sum_{n=0}^{2N-1} x_2(n)e^{-j2\pi k\frac{n}{2N}} = \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)e^{-j2\pi k\frac{n}{2N}} + \sum_{n=N}^{2N-1} x(2N-1-n)e^{-j2\pi k\frac{n}{2N}}\right) \\
&= \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)\left(e^{-j2\pi k\frac{n}{2N}} + e^{-j2\pi k\frac{(2N-1-n)}{2N}}\right)\right) \\
&= \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)\left(e^{-j2\pi k\frac{n}{2N}} + e^{j2\pi k\frac{(1+n)}{2N}}\right)\right) \\
&= \frac{1}{2N}\left(\sum_{n=0}^{N-1} x(n)e^{j\pi k\left(\frac{1}{2N}\right)}\left(e^{-j\pi k(\frac{2n+1}{2N})} + e^{j\pi k(\frac{2n+1}{2N})}\right)\right) \\
&= \frac{e^{j\pi k\left(\frac{1}{2N}\right)}}{N}\left(\sum_{n=0}^{N-1} x(n)\cos\left(\pi k\,(\frac{2\mathrm{n}+1}{2N})\right)\right)
\end{aligned}
$$

The summation is just on the first N values, interesting, no ?

Discrete Cosine Transform

Discrete Cosine Transform

- The 1-D discrete cosine transform is defined as :

$$F(k) = \alpha(k) \sum_{n=0}^{N-1} x(n) \cos\left[\frac{(2n+1)k\pi}{2N} \right]$$

$$k = 0, \cdots, N-1$$

$$\alpha(k) = \begin{cases} \sqrt{\dfrac{1}{N}} & ; \quad k = 0 \\ \sqrt{\dfrac{2}{N}} & ; \quad k \neq 0 \end{cases}$$

This normalization parameters makes the DCT matrix orthonormal

Discrete Cosine Transform

- DCT express a signal in terms of a sum of cosines with different frequencies and amplitudes.
- The DCT of real function, $F(k)$, is real
- In general, the DCT is not the real part of the DFT

- A is real and orthogonal
  - rows of A form orthonormal basis
  - A is not symmetric!

Inverse DCT

- Similarly, the Inverse DCT (IDCT) is defined as

$$x(n) = \sum_{k=0}^{N-1} \alpha(k) F(k) \cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

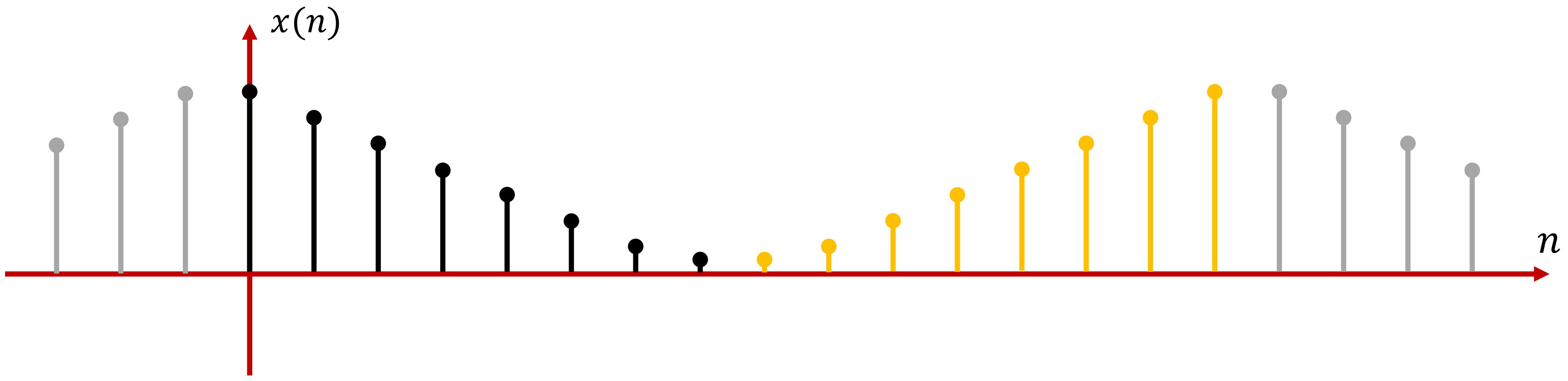
$$n = 0, \cdots, N-1$$

$$\alpha(0) = \sqrt{\frac{1}{N}}$$

$$\alpha(n) = \sqrt{\frac{2}{N}}$$

Periodicity of the DCT

- The DCT implies a periodic extension of the original function.
- The DCT implies the boundary conditions: $x(n)$ is even around $n = -1/2,$ this different from the boundary condition of the DFT



1-D DCT in matrix notations

- The kernel of the 1-D discrete cosine transform (DCT-II) is :

$$
\begin{aligned}
F(k) &= \alpha(k)\sum_{n=0}^{N-1} x(n)\cos\left[\frac{(2n+1)k\pi}{2N}\right] \\
&= \sum_{n=0}^{N-1} x(n)\alpha(k)\cos\left[\frac{(2n+1)k\pi}{2N}\right] \\
&= \sum_{n=0}^{N-1} x(n)w(k,n)
\end{aligned}
$$

kernel

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

1-D DCT in matrix notations

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ \vdots \\ F(N-1) \end{bmatrix} = \begin{bmatrix} w(0,0) & w(0,1) & \cdots & \cdots & w(0,N-1) \\ w(1,0) & w(1,1) & & & \\ \vdots & & \cdots & & \vdots \\ \vdots & & & \cdots & \vdots \\ w(N-1,0) & w(N-1,1) & \cdots & \cdots & w(N-1,N-1) \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}$$

$$\mathbf{F} = \mathbf{TX} \quad \Rightarrow \quad \mathbf{X} = \mathbf{T}^{-1}\mathbf{F}$$

The transformation matrix , *T* , is *NxN* matrix

Mat. Ex.

2-D DCT

- The two-dimensional DCT (2D-DCT) could be written also as :

$$F(k,l) \quad = \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

$$k,l = 0,\ldots,N-1$$

$$\alpha(0) = \sqrt{\frac{1}{N}} \quad \alpha(k) = \sqrt{\frac{2}{N}}$$

n row index, m column,
k row index, l column

2-D DCT

- The two-dimensional DCT (2D-DCT) could be written also as :

$$
\begin{aligned}
F(k,l) &= \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]\cos\left[\frac{(2n+1)k\pi}{2N}\right] \\
&= \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\left(\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]\right)\cos\left[\frac{(2n+1)k\pi}{2N}\right]
\end{aligned}
$$

$$k,l = 0,\dots,N-1$$

$$\alpha(0) = \sqrt{\frac{1}{N}} \quad \alpha(k) = \sqrt{\frac{2}{N}}$$

n row index, m column,
k row index, l column

2-D DCT

- The two-dimensional DCT (2D-DCT) could be written also as :

$$F(k,l) = \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

$$= \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\boxed{\left(\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]\right)}\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

What is this term ?

$$k,l = 0,\ldots,N-1$$

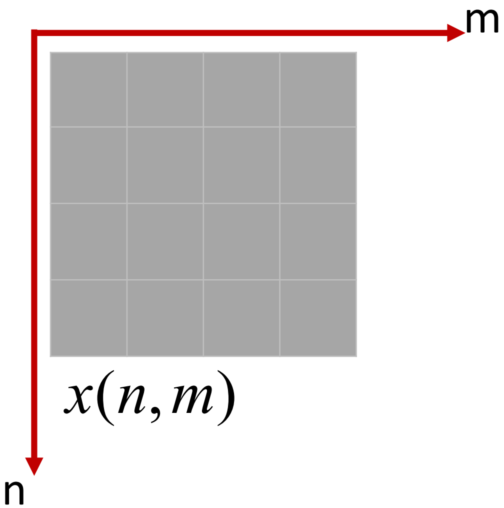
$$\alpha(0) = \sqrt{\frac{1}{N}} \quad \alpha(k) = \sqrt{\frac{2}{N}}$$

n row index, m column,
k row index, l column

2-D DCT

- The two-dimensional DCT is obtained applying the 1-D transform to the columns (rows) and then to the rows (columns) independently

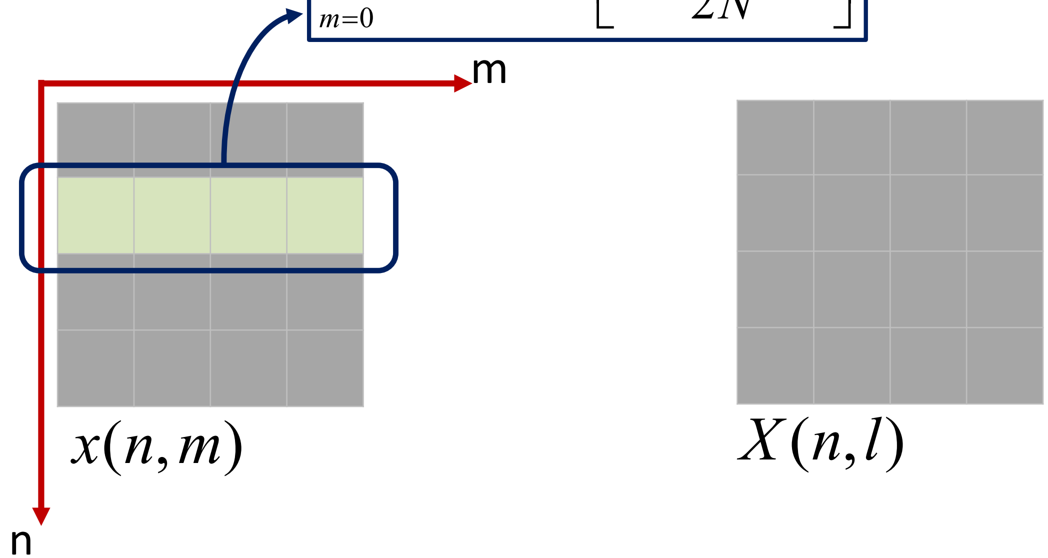
2-D DCT



$x(n,m)$

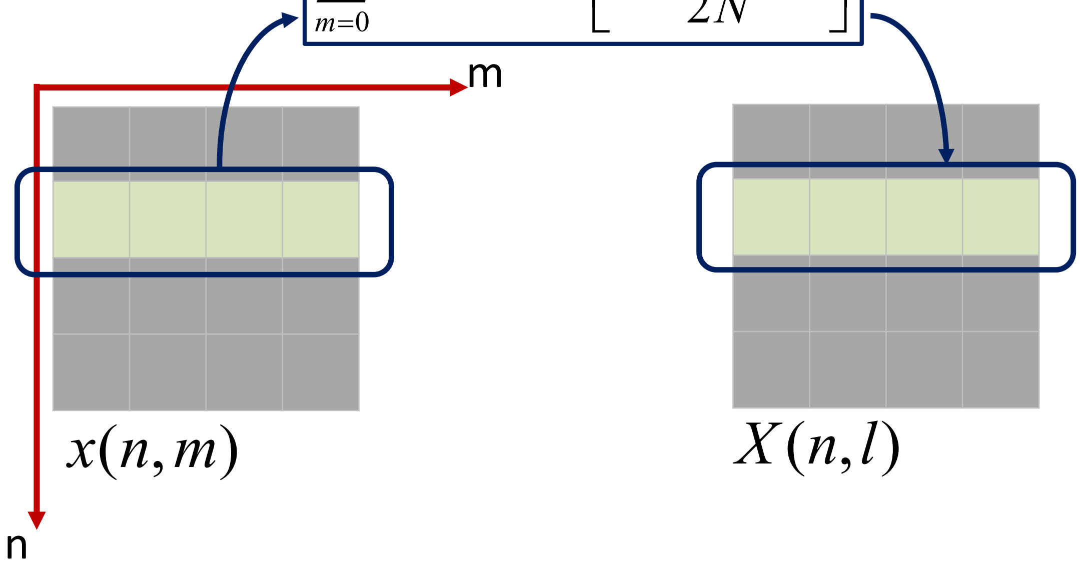
2-D DCT

$$\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]$$

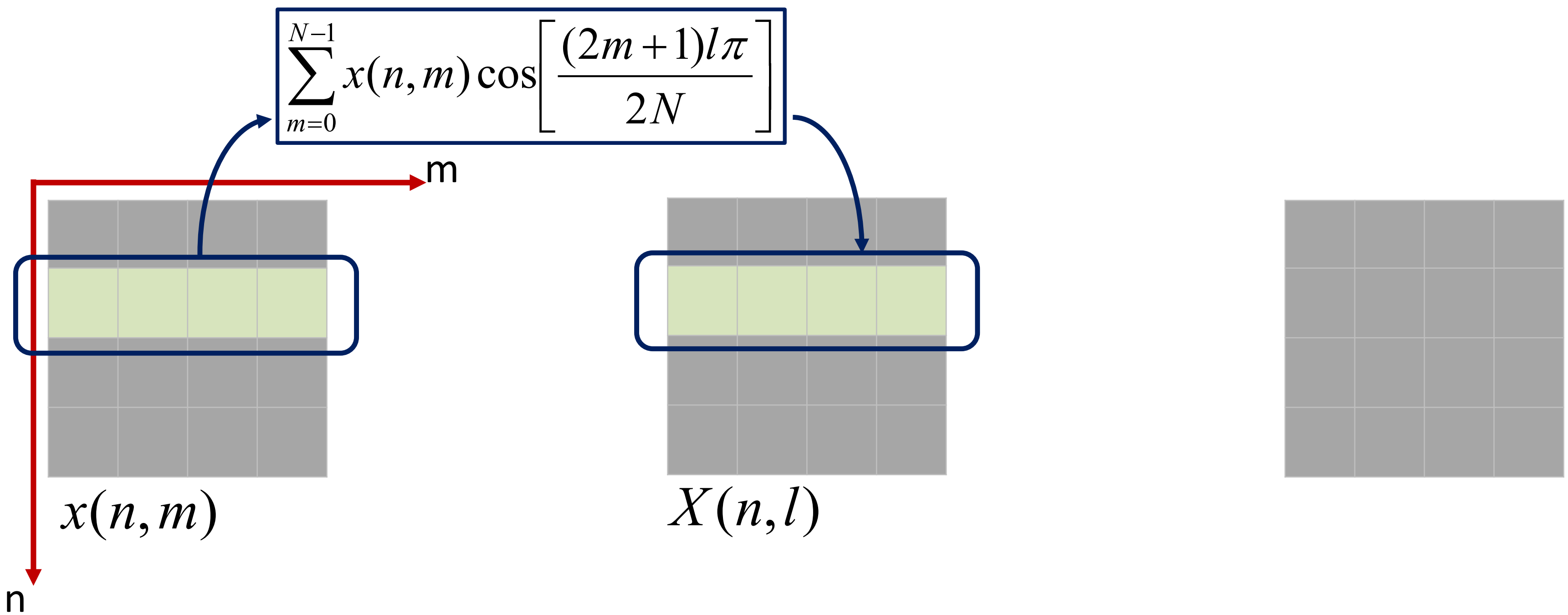


2-D DCT

$$\sum_{m=0}^{N-1} x(n,m)\cos\left[\frac{(2m+1)l\pi}{2N}\right]$$

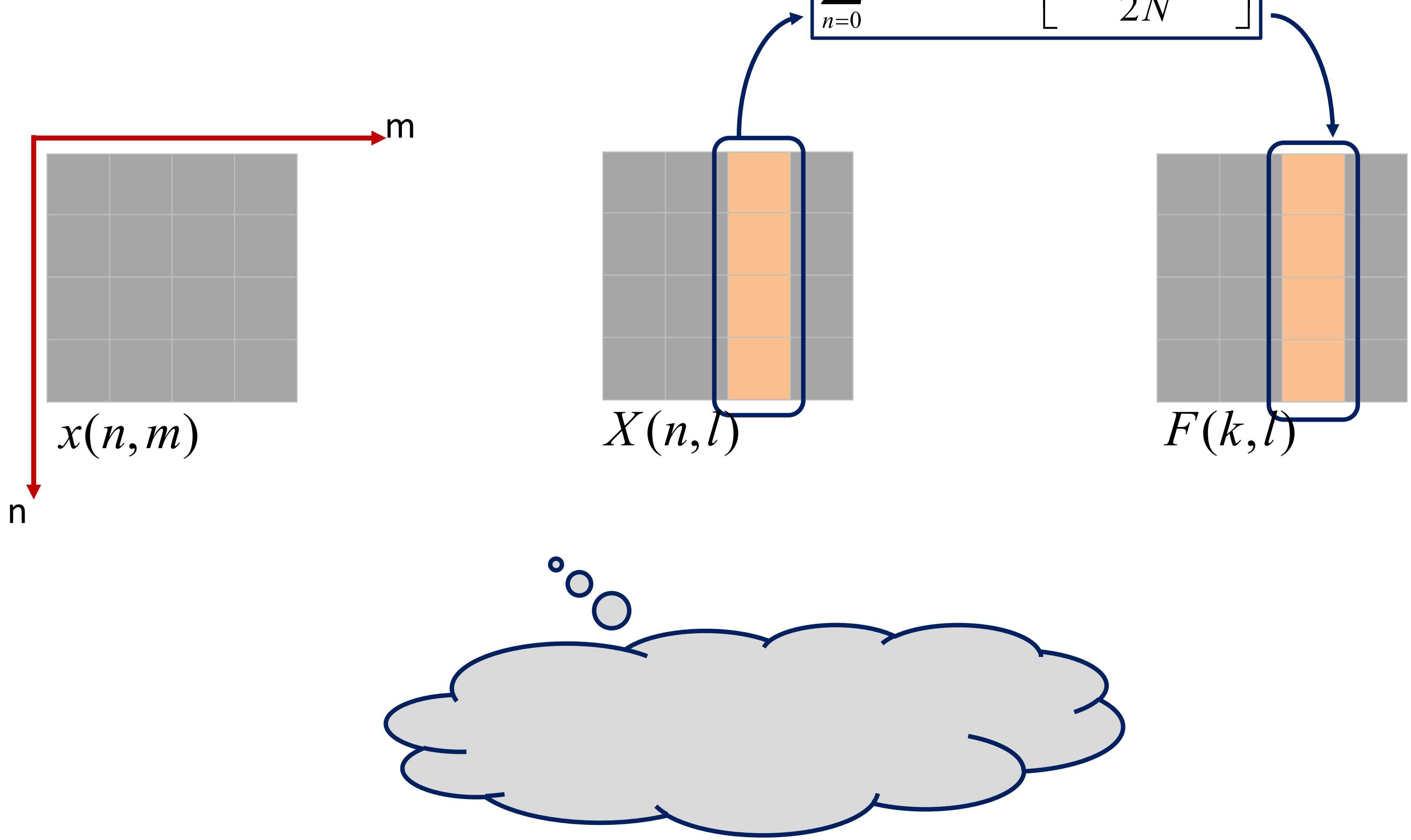


2-D DCT



2-D DCT

$$\boxed{\sum_{n=0}^{N-1} X(n,l)\cos\left[\frac{(2n+1)k\pi}{2N}\right]}$$



2-D DCT

- The two-dimensional DCT is obtained applying the 1-D transform to the columns (rows) and then to the rows (columns) independently
- This kind of transform is called separable transform

2-D DCT

- In matrix notation
  - Applying the DCT to the columns :

$$\mathbf{F}_{tmp} = \mathbf{TX}$$

  - Applying the DCT to the rows of the resulting image is :

$$\mathbf{F} = \mathbf{F}_{tmp}\mathbf{T}^{\mathbf{T}}$$

  - The two-dimensional DCT (2D-DCT) could be written also as :

$$\mathbf{F} = \mathbf{TXT}^{\mathbf{T}}$$

Inverse 2-D DCT

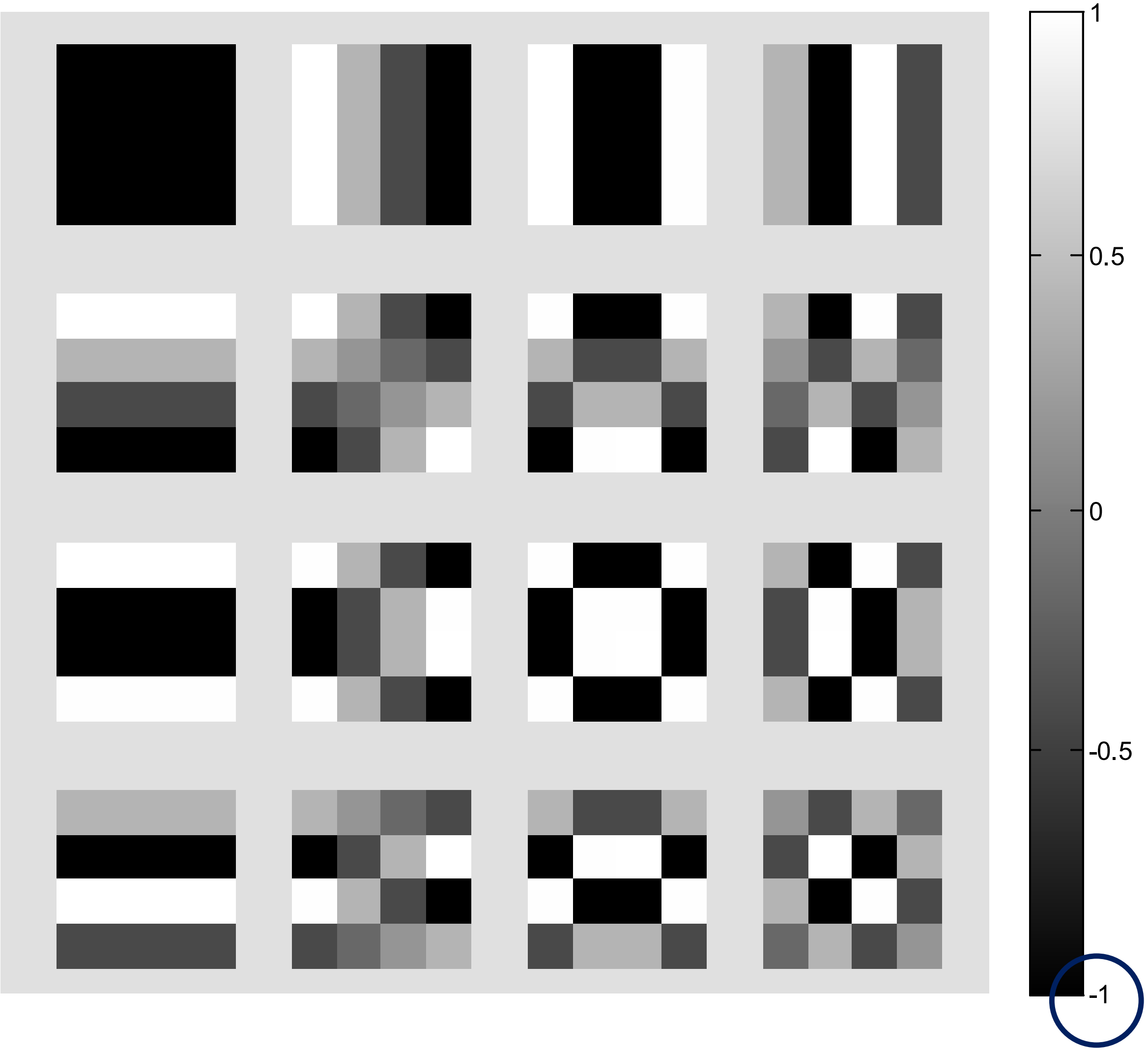
- Analogously, the inverse 2-D transform is

$$x(n,m)=\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\alpha(k)\alpha(l)F(k,l)\cos\left[\frac{(2n+1)k\pi}{2N}\right]\cos\left[\frac{(2m+1)l\pi}{2N}\right]$$

$$n,m=0,\ldots,N-1$$

$$\alpha(0)=\sqrt{\frac{1}{N}}\quad \alpha(i)=\sqrt{\frac{2}{N}}\quad i\neq 0$$

2D-DCT basis functions



code

Basis functions of 4X4 DCT

When it is applied to an 4x4 image, it yields an 4x4 matrix of weighted values corresponding to how much of each basis function is present in the image

An 4x4 image that just contains one shade of gray will yield only a weighted value for the upper left hand DCT basis function (which has no frequencies in the x or y direction).

Interpretation of DCT basis functions

- The DCT helps separate the image into spectral sub-bands of differing importance with respect to the image's visual quality.

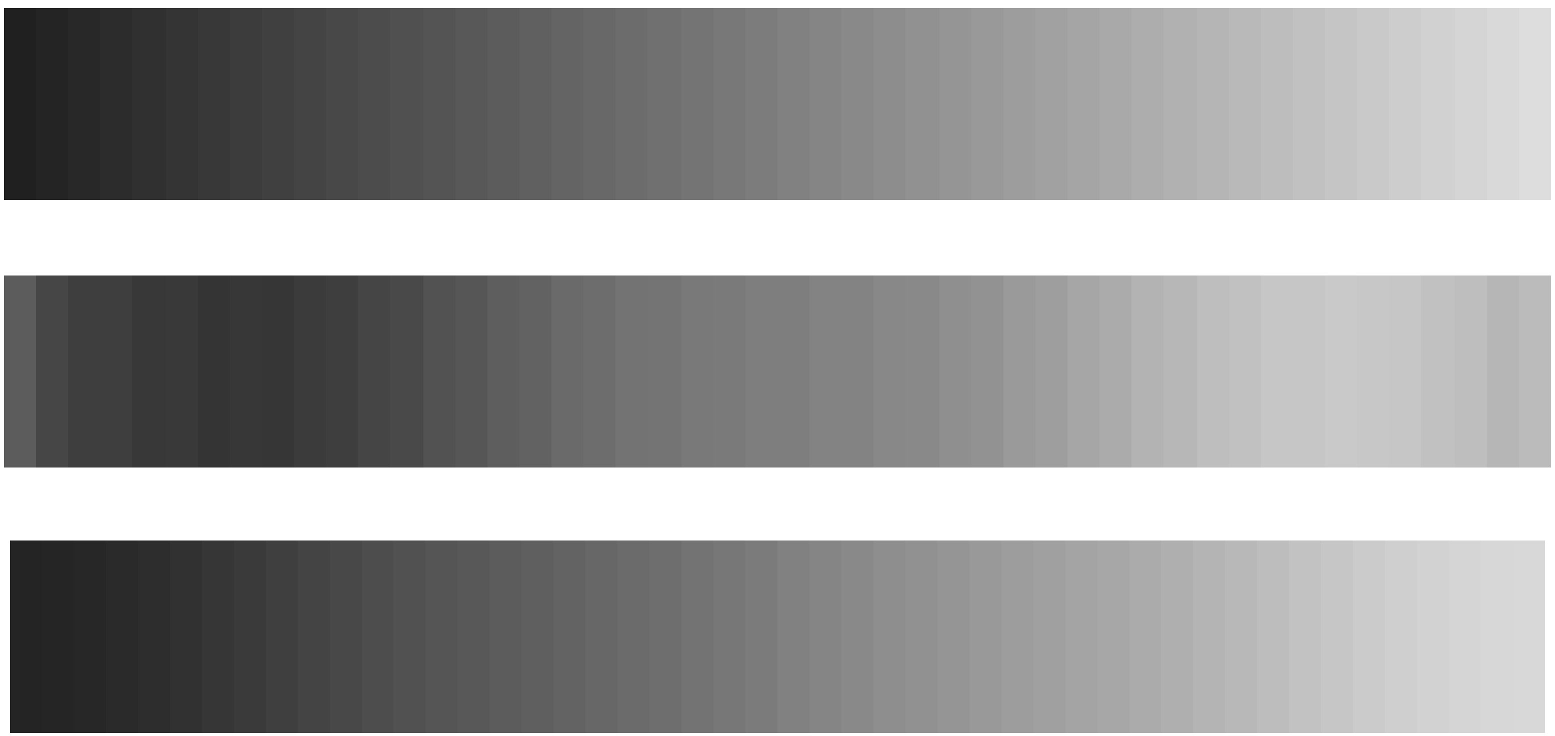
- Any grey-scale $N \times N$ pixel block can be fully represented by a weighted sum of the N^2 basis functions → the DCT coefficients acting as weights for these blocks.

Interpretation of DCT basis functions

- The top-left basis function represents zero spatial frequency (*DC coefficient*)
- Along the top row the basis functions have increasing horizontal spatial frequency content.
- Down the left column the functions have increasing vertical spatial frequency content.

Why DCT not FFT for image applications ?

- Zeroing 88% of high frequencies, i.e., keeping only 12% of low frequencies

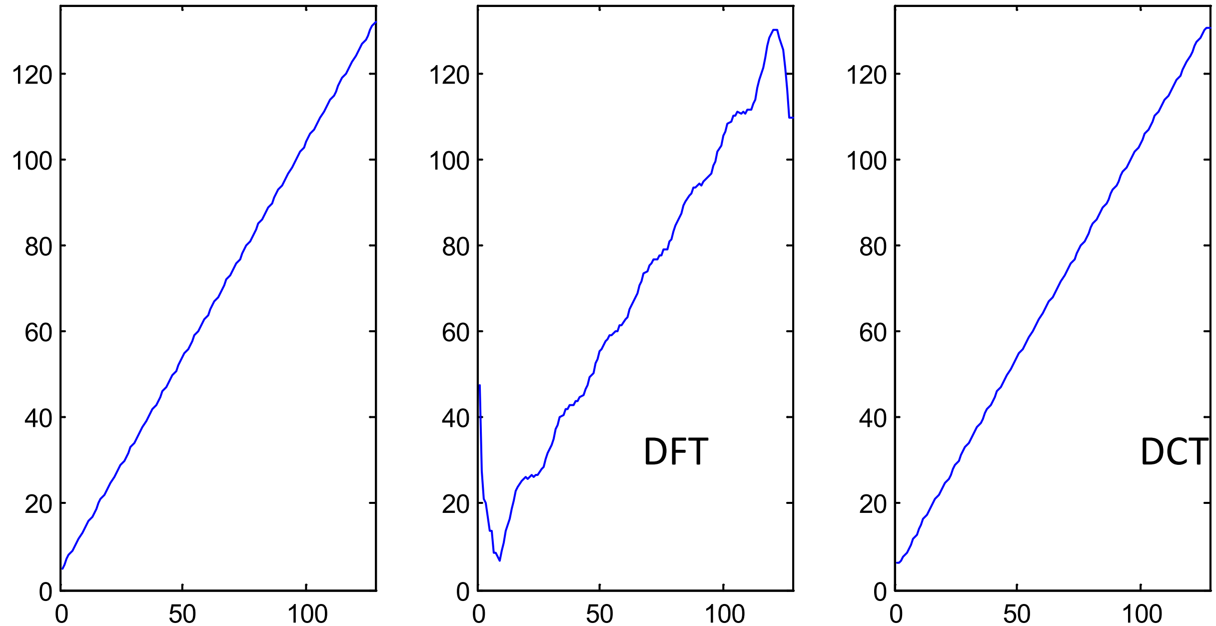


DFT

DCT

Why DCT not FFT for image applications ?

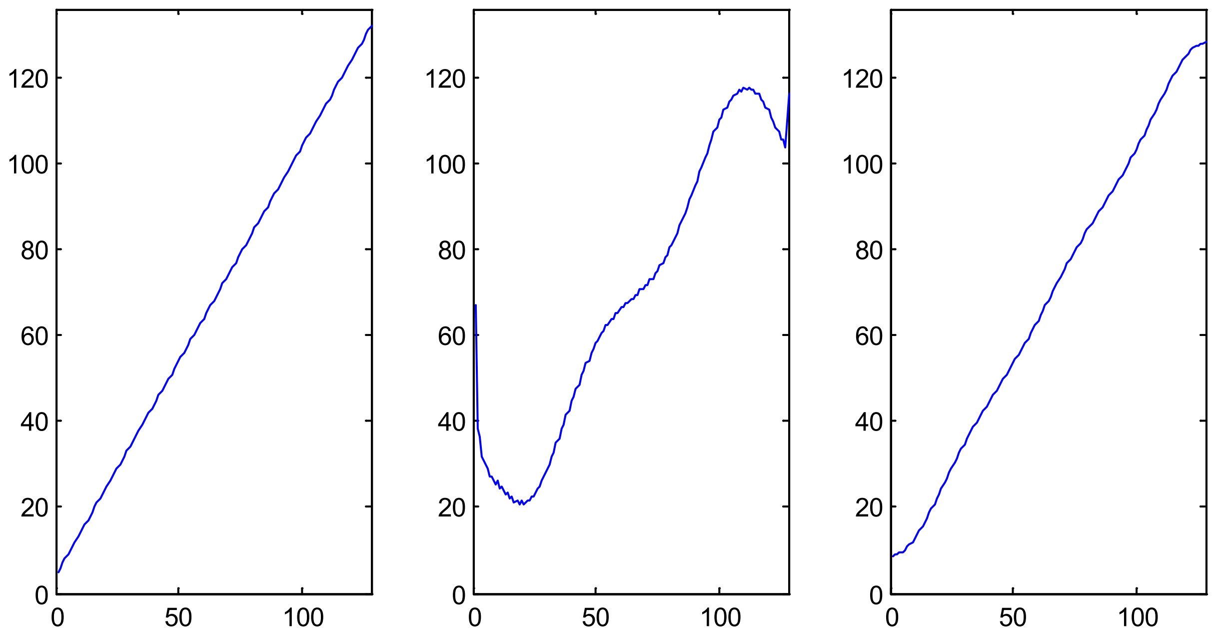
- Keeping only 12% of low frequencies



  - DCT can approximate boundary discontinuity well with fewer coefficients (Discontinuities require more high frequency components to represent them)
  - Blocking artifacts less pronounced

Why DCT not FFT for image applications ?

- Keeping only 5% of low frequencies



  - DCT can approximate boundary discontinuity well with fewer coefficients (Discontinuities require more high frequency components to represent them)
  - Blocking artifacts less pronounced

DCT applications

- The DCT is a popular transform used by the JPEG (Joint Photographic Experts Group) image compression standard for lossy compression of images.
- Since it is being so famous in the field of image compression in nowaday literature is often referring to the DCT used in JPEG as the JPEG-DCT.

Self-Reading

Hadamard transform

Hadamard transform

- It uses basis vectors constituted of +1 and –1
- Due to its simplicity it is used in variety of applications.
- Simple recursive relationship holds for generating the transformation matrices starting from the lower order one (N=2)

Hadamard transform

- Hadamard matrix of order N=2:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- Hadamard matrix of order $2N$:

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

Hadamard transform

- Hadamard matrix of order N=4:

$$H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} + & + & + & + \\ + & - & + & - \\ + & + & - & - \\ + & - & - & + \end{bmatrix}$$

Hadamard transform

- Kernel of the Hadamard transform (in the case N=8):

$$H_8 = \begin{bmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{bmatrix} = \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & - & + & - & + & - & + & - \\ + & + & - & - & + & + & - & - \\ + & - & - & + & + & - & - & + \\ + & + & + & + & - & - & - & - \\ + & - & + & - & - & + & - & + \\ + & + & - & - & - & - & + & + \\ + & - & - & + & - & + & + & - \end{bmatrix}$$

$$U = HV$$

- *V, U,* vectors of original and transformed coefficients respectively

Hadamard transform

- Observations:
  - The transformation has no multiplications
  - If the input signal is real then the transformed signal is also real, because Hadamard matrix has only two real values, +1 or -1.
  - The rows and columns of the Hadamard matrix form an orthogonal set

Image Filters

(Low-pass) Filtering in the Fourier Space

We thus create a new version of the image in Fourier space by computing

$$G(u,v) = H(u,v)F(u,v)$$

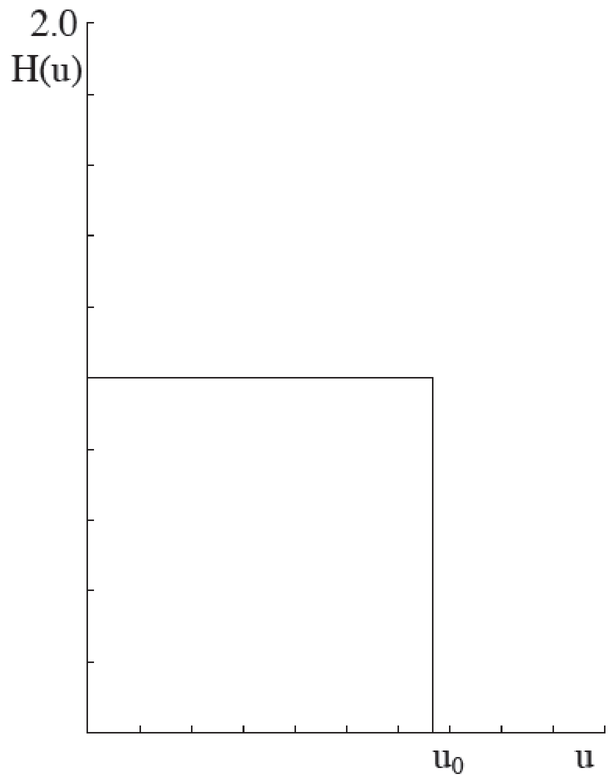
where:

- $F(u,v)$ is the Fourier transform of the original image,
- $H(u,v)$ is a filter function, designed to reduce high frequencies and
- $G(u,v)$ is the **Fourier transform of the improved image**.
- Inverse Fourier transform $G(u,v)$ to get $g(x,y)$ our **improved image**

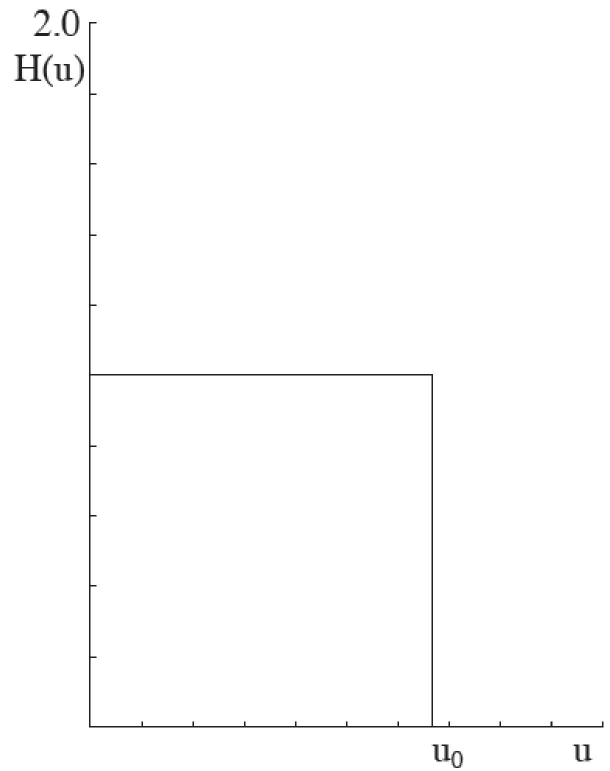
Note: Discrete Cosine Transform approach identical, sub. FT with DCT

Ideal Low-Pass Filter

The simplest sort of filter to use is an *ideal low-pass filter*, which in one dimension appears as :



Ideal Low-Pass Filter (Cont.)



This is a top hat function which is 1 for u between 0 and u_0, the *cut-off frequency*, and zero elsewhere.

- So All frequency space information above u_0 is thrown away, and all information below u_0 is kept.
- A **very simple** computational process.

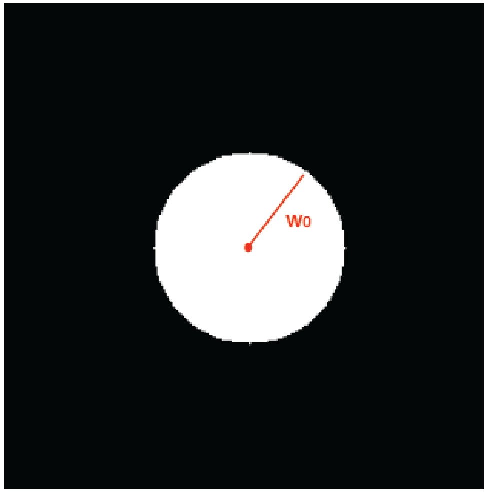
Ideal 2D Low-Pass Filter

The two dimensional analogue of this is the function

$$H(u,v) = \begin{cases} 1 & \text{if } \sqrt{u^2+v^2} \leq w_0 \\ 0 & \text{otherwise,} \end{cases}$$

where w_0 is now the cut-off frequency.

Thus, all frequencies inside a radius w_0 are kept, and all others discarded.



Ideal Low-Pass Filter Example 1 MATLAB Code

low pass.m:

```
% Create a white box on a black background image
M = 256; N = 256;
image = zeros(M,N)
box = ones(64,64);
%box at centre
image(97:160,97:160) = box;

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F)));
```

Ideal Low-Pass Filter Example 1 MATLAB Code (Cont.)

```
%compute Ideal Low Pass Filter
u0 = 20; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);

% display
figure(3);
imshow(fftshift(H));

% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));

% Show Result
figure(4);
imshow(g);
```

Not So Ideal Low-Pass Filter?

The problem with this filter is that as well as the noise:

- In audio: plenty of other high frequency content
- In Images: edges (places of rapid transition from light to dark) also significantly contribute to the high frequency components.

Thus an ideal low-pass filter will tend to *blur* the data:

- High audio frequencies become muffled
- Edges in images become blurred.

The lower the cut-off frequency is made, the more pronounced this effect becomes in *useful data content*

Low-Pass Butterworth Filter

Another filter sometimes used is the *Butterworth low pass filter*.

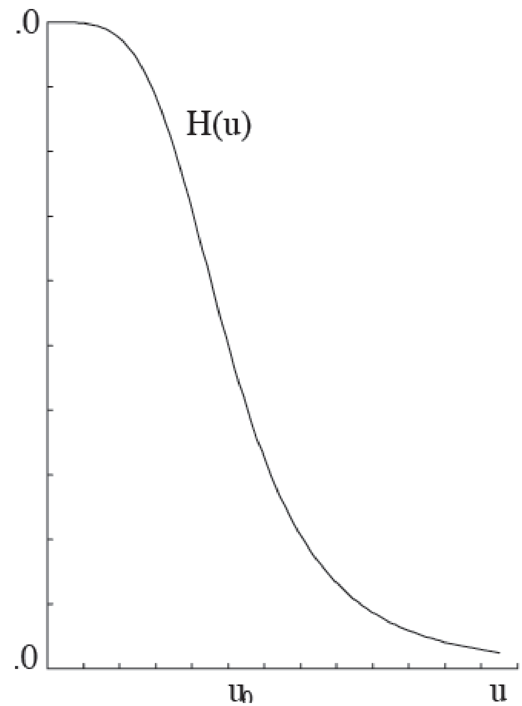
In the 2D case, $H(u,v)$ takes the form

$$H(u,v) = \frac{1}{1 + [(u^2+v^2)/w_0^2]^n},$$

where n is called the **order** of the filter.

Low-Pass Butterworth Filter (Cont.)

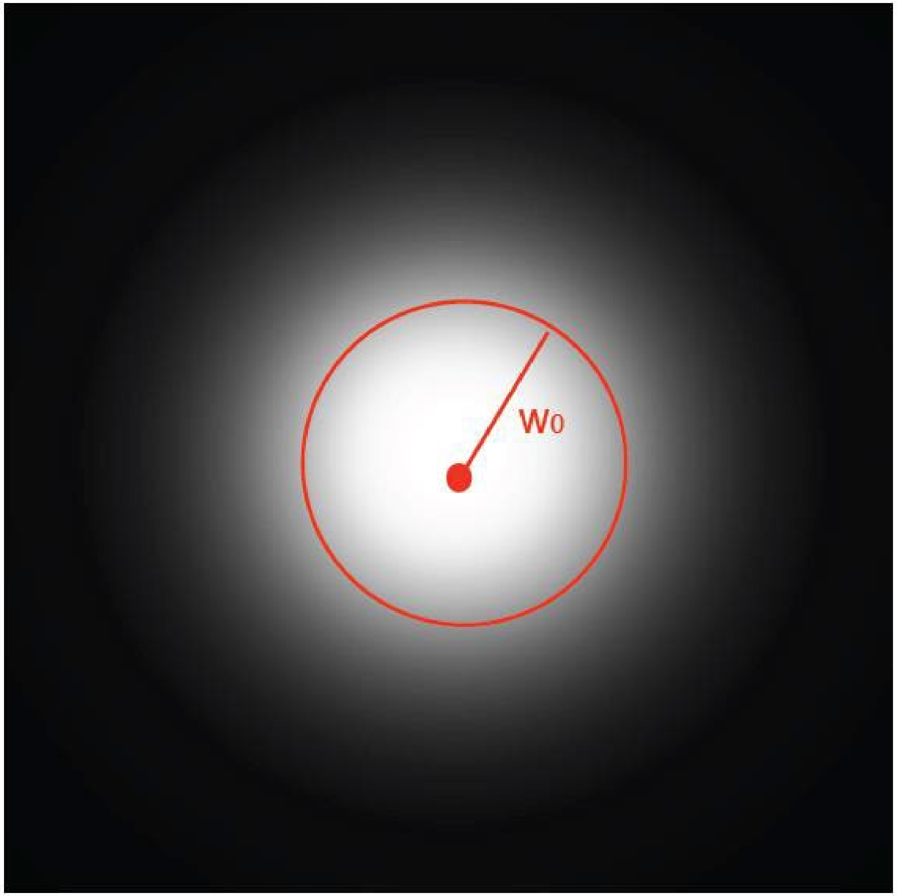
This keeps some of the high frequency information, as illustrated by the second order one dimensional Butterworth filter:



Consequently reduces the blurring.

Low-Pass Butterworth Filter (Cont.)

The 2D second order Butterworth filter looks like this:



Butterworth Low-Pass Filter Example 1 MATLAB Code

butterworth.m:

```
% Load Image and Compute FFT as in Ideal Low Pass Filter
% Example 1
.......
% Compute Butterworth Low Pass Filter
u0 = 20; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

for i = 1: M
    for j = 1:N
      %Apply a 2nd order Butterworth
      UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j))/(u0*u0));
      H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```

Thanks