**FOURIER TRANSFORM**

**DFT as a Co-ordinate Transform**

\[
F[u, v] = \frac{1}{\sqrt{WH}} \sum_{n_x=0}^{W-1} \sum_{n_y=0}^{H-1} S_{uv}[n_x, n_y] X[n_x, n_y]
\]

where each \( S_{uv} \) can be thought of as a different (complex-valued) image:

\[
S_{uv}[n_x, n_y] = \frac{1}{\sqrt{WH}} \exp\left(j 2\pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right)
\]

and \( \overline{S_{uv}} \) denotes the complex-conjugate of \( S_{uv} \).

**FOURIER TRANSFORM**

- \( F[u, v] \) and \( X[n_x, n_y] \) are both 2D array of the same size \( W \times H \).
- \( F \) is complex-valued (while \( X \) is typically real-valued)
- These equations are linear. So both the Fourier transform and its inverse are linear operations.
- But each \( F[u, v] \) depends on values of \( X[n_x, n_y] \) at ALL locations (not local like a convolution).
- Note that the \( \exp() \) expressions in both are similar—one has a negative sign inside the \( \exp \), indicating a complex conjugate.
- But, in one, we hold \((u, v)\) fixed and sum over \((n_x, n_y)\). In the other, vice-versa.
FOURIER TRANSFORM

DFT as a Co-ordinate Transform

\[ F[u, v] = \frac{1}{\sqrt{WH}} \langle S_{uv}, X \rangle \]

where each \( S_{uv} \) can be thought of as a different (complex-valued) image:

\[ S_{uv}[n_x, n_y] = \frac{1}{\sqrt{WH}} \exp\left(j 2 \pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

\( F[u, v] \) is the inner-product between \( X \) and \( S_{uv} \) (scaled by \( \sqrt{WH} \)).

\[ S_{uv} = \]

**Property**: \( \langle S_{uv}, S_{u'v'} \rangle = 1 \) if \( u' = u \) & \( v' = v \), and 0 otherwise.

Inverse-DFT:

\[ X[n_x, n_y] = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F[u, v] \exp\left(j 2 \pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

\( X \) is a weighted sum of the \( S_{uv} \) images, weights are given by \( \sqrt{WH} F[u, v] \).

**FOURIER TRANSFORM**

DFT as a Co-ordinate Transform

\[ F[u, v] = \frac{1}{\sqrt{WH}} \langle S_{uv}, X \rangle \]

where each \( S_{uv} \) can be thought of as a different (complex-valued) image:

\[ S_{uv}[n_x, n_y] = \frac{1}{\sqrt{WH}} \exp\left(j 2 \pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

\( F[u, v] \) is the inner-product between \( X \) and \( S_{uv} \) (scaled by \( \sqrt{WH} \)).

**Property**: \( \langle S_{uv}, S_{u'v'} \rangle = 1 \) if \( u' = u \) & \( v' = v \), and 0 otherwise.

Inverse-DFT:

\[ X[n_x, n_y] = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F[u, v] \exp\left(j 2 \pi \left( \frac{u n_x}{W} + \frac{v n_y}{H} \right) \right) \]

\( X \) is a weighted sum of the \( S_{uv} \) images, weights are given by \( \sqrt{WH} F[u, v] \).
DFT as a Co-ordinate Transform

\[ F[u, v] = \frac{1}{\sqrt{WH}} \langle S_{av}, X \rangle, \quad X = \sqrt{WH} \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} F[u, v] S_{av} \]

\[ S = \begin{bmatrix} S_{00} & \cdots & S_{0,\text{max}} \\ \vdots & \ddots & \vdots \\ S_{\text{max,0}} & \cdots & S_{\text{max,\text{max}}} \end{bmatrix} \]

\[ X[n_x, n_y] = S^* \cdot X \]

The matrix \( S \) is a \( WH \times WH \) matrix with each column a different \( S_{av} \).

\[ \langle S_{av}, S'_{av} \rangle = 1 \text{ if } u' = u \text{ & } v' = v, \text{ and } 0 \text{ otherwise.} \]

So, \( SS^* = S^*S = I \Rightarrow S^{-1} = S^* \)

- This means \( S \) is a unitary matrix.
- Multiplication by \( S \) is a co-ordinate transform:
  - \( X \) are the co-ordinates of a point in a \( WH \) dimensional space.
  - Multiplication by \( S^* \) changes the ‘co-ordinate system’.
  - In the new co-ordinate system, each ‘dimension’ now corresponds to frequency rather than location.
  - \( S \) is a length-preserving matrix (\( ||S^*X||^2 = ||X||^2 \)).
  - It does rotations or reflections (in \( WH \) dimensional space).
FOURIER TRANSFORM

Zero-centered Co-ordinates for frequencies \((u,v)\)

Reconstruct with only these frequency components

Reconstruct with only these frequency components
FOURIER TRANSFORM

$X$  $|F|^2$  $\angle F$

Reconstruct with only these frequency components

FOURIER TRANSFORM

$X$  $|F|^2$  $\angle F$

Reconstruct with only these frequency components

FOURIER TRANSFORM

$X$  $|F|^2$  $\angle F$

FOURIER TRANSFORM

$X$  $|F|^2$  $\angle F$
The FT gave us a different representation for images. Decomposing image into different frequency 'components'.

What else?
CONVOLUTION THEOREM

Convolution in "matrix" form

\[ Y[n_x, n_y] = \sum_{n_x'} \sum_{n_y'} k[n_x', n_y'] X[n_x - n_x', n_y - n_y'] \]
**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

Convolutions in "matrix" form

\[
\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}
\]

\[ Y[n_x, n_y] = \sum_{n_x'} \sum_{n_y'} k[n_x', n_y'] X[n_x - n_x', n_y - n_y'] \]

**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

\[ Y = X * k \Rightarrow Y = A_k X \]

\( A_k \) is not square for valid / long convolution.

**Question**

Let \( Y = A_k X \) correspond to \( Y = X * k \). Now, let \( X' = A_k^T Y \). How is \( X' \) related to \( Y \) by convolution ?

**What operation does \( A_k^T \) represent ?**

A: Full convolution with \( k[-n_x, -n_y] \) (flipped version of \( k \))

\[
Y = X * k \Rightarrow Y = X A_k
\]

\[
Y = X A_k
\]

\[
Y = X A_k
\]

\[
Y = X A_k
\]

\[
Y = X A_k
\]

Now if we consider the square \( A_k \) matrix corresponding to ‘same’ convolution with circular padding:

\[
X[W + n_x, n_y] = X[n_x, n_y]
\]

\[
X[n_x, -n_y] = X[n_x, H - n_y]
\]

Then, \( A_k \) is diagonalized by the Fourier Transform !

\[
A_k = S D_k S^t
\]

- Here, \( D_k \) is a diagonal matrix.
- The above equation holds for every \( A_k \).
- You get different diagonal matrices \( D_k \).
- But \( S \) is the diagonalizing basis for all kernels.
- In the Fourier co-ordinate system, convolution is a ‘point-wise’ operation !

\[
Y = A_k X = S D_k S^t X \Rightarrow (S^t Y) = D_k (S^t X)
\]

\[
Y = X * k = \sum_u \sum_v F[u, v] S_{uv} \ast k = \sqrt{WH} \sum_u \sum_v (F[u, v] d_{uv, k}) S_{uv}
\]

**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

**CONVOLUTION THEOREM**

Why does this happen ?

- \( X = \sqrt{WH} \sum_u \sum_v F[u, v] S_{uv} \)
- \( Y = X * k = \sqrt{WH} \sum_u \sum_v F[u, v] S_{uv} \ast k \) (by linearity / distributivity)
- \( \langle S_{uv} \ast k[n] \rangle = \sum_{n'} k[n'] S_{uv}[n - n'] \)
- \( S_{uv}[n - n'] \), assuming circular padding, is also a sinusoid with the same frequency \( (u, v) \) and magnitude, but different phase.
- Multiplying by \( k[n'] \) changes the magnitude, but frequency still the same.
- Adding different sinusoids of the same frequency gives you another sinusoid of the same frequency.
- \( \langle S_{uv} \ast k[n, n] \rangle = d_{uv, k} S_{uv}[n, n], \) where \( d_{uv, k} \) is some complex scalar.

Sinusoids are eigen-functions of convolution

\[
Y = X * k = \sqrt{WH} \sum_u \sum_v F[u, v] d_{uv, k} S_{uv}
\]
What's more, the diagonal elements of $D_k$ are the Fourier transform of $k$ (assuming it's of size $W \times H$).

$$D_k = \text{diag} \left( \frac{1}{\sqrt{WH}} S^* k \right)$$

This is the convolution theorem.

- Computational advantage for performing (and inverting!) convolution, albeit under circular padding.
- Good way of analyzing what a kernel is doing by looking at its Fourier transform.

---

Doing Convolutions in the Fourier Domain:
- DFT, Point-wise multiply with FT of kernel, Inverse DFT
- Need to keep in mind some padding / size issues.

---

Kernel has to be the same size as the image.

1. Zero-pad
CONVOLUTION THEOREM

Kernel has to be the same size as the image.

1. Zero-pad
2. Circularly shift to center at (0,0)

- From same circular, you can always get 'valid' by cropping.
- To get full / same with zero-padding, pad your original image first.
CONVOLUTION THEOREM

Kernel / Fourier Transform (magnitude) Pairs

Gaussian Kernels: Low Pass (attenuate higher frequencies)
Larger spatial support: smaller Fourier support.

Gaussian Derivatives: Band-pass

For more in-depth coverage:
Szeliski Sec 3.4
SCALE & ALIASING

“Resize” Images

SCALE & ALIASING

“Resize” Images

SCALE & ALIASING

“Resize” Images

SCALE & ALIASING

“Resize” Images

[Source: Wikipedia]
SCALE & ALIASING

"Resize" Images

(W/2) x (H/2)

[Source: Wikipedia]

"Aliasing"

SCALE & ALIASING

If you write it out, you see the higher freq. components get folded into lower freq.

Remember, in the two cases \( F_{(u,v)} \) is defined with respect to different width and height \( W_k \) and \( H_k \), and for different ranges of \((u,v)\).

SCALE & ALIASING

If you write it out, you see the higher freq. components get folded into lower freq.

Remember, in the two cases \( F_{(u,v)} \) is defined with respect to different width and height \( W_k \) and \( H_k \), and for different ranges of \((u,v)\).

Make sure there are no high frequencies before sub-sampling!
Make sure there are no high frequencies before sub-sampling!

Low-pass filter, i.e., smooth image before sub-sampling.

“Resize” images

- Need to hallucinate missing information.
- Lots of research (super-resolution).

Sometimes the camera itself makes aliased measurements: if spatial sensitivity is low at edges of pixel.
"Resize" Images

- Need to hallucinate missing information.
- Lots of research (super-resolution).
- Simplest Approach: Nearest neighbor

\[ Y[n] = X[\text{round}(n/2)] \]

For up-sampling by 2 in 1-D, missing values are just the average of the left and right present values.

- Simplest Approach: (Bi) Linear Interpolation
Can achieve this by filling with zeros, and convolution with a 3x3 kernel.

Can achieve this by filling with zeros, and convolution with a 3x3 kernel.
Can achieve this by filling with zeros, and convolution with a 3x3 kernel.

Can achieve this by filling with zeros, and convolution with a 3x3 kernel.
**SCALE & ALIASING**

Convolution, in the most general case, takes \( O(n \times n) \) time.

- \( n_1 = W \times H_1, n_2 = W \times H_2 \).

Convolution in the frequency domain:
- FFT, point-wise multiply, Inverse FFT
- FFT/IFFT complexity is \( O(n \times \log_2 n) \) (Most efficient for power of 2 image size)
- May be worth it for large kernels
- Or same image convolved with many different kernels

Can achieve this by filling with zeros, and convolution with a 3x3 kernel.
Efficient Computation

Separable Kernels

\[ G[n_x, n_y] \propto \exp \left( -\frac{n_x^2 + n_y^2}{2\sigma^2} \right) = G_x[n_x]G_y[n_y] \]

- \( x \) - and \( y \) - derivatives of Gaussian also separable.
- Realize that \( k[n_x, n_y] = k_x[n_x], k[n_y] = k_y \ast k_y \).
  This is by interpreting \( k_x \) and \( k_y \) as having size \( W_k \times 1 \) and \( 1 \times H_k \).
- So \( X \ast k = X \ast (k_x \ast k_y) = (X \ast k_x) \ast k_y \). This takes \( W_k + H_k \) operations instead of \( W_k H_k \).
- Often if a kernel itself isn’t separable, it can be sometimes expressed as a sum of separable kernels.
  - E.g., Unsharp Mask: \((1 + \alpha)\delta - \alpha G_\sigma \) (don’t combine!)