CSE 559A: Computer Vision

Fall 2020: T-R: 11:30-12:50pm @ Wrighton 300 / Zoom

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Adith Boloor, Patrick Williams

http://www.cse.wustl.edu/~ayan/courses/cse559a/

October 1, 2020
FOURIER TRANSFORM

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

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

- \(F[u, v]\) 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 \(\exp(\cdot)\) 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.
DFT as a Co-ordinate Transform

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

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

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 \( \sqrt{WH} \))

For \( x, y \in \mathbb{C}^n \), \( \langle x, y \rangle = \sum_i x_i y_i \)

- For real vectors, \( \langle x, y \rangle = x^T y \).
- For complex vectors, \( \langle x, y \rangle = x^* y \)
  - \( x^* \) is the Hermitian of \( x \): transpose AND conjugate
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 \( \sqrt{WH} \))
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:

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

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

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

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

"Frequency" Locations
Stacked to form Vector

\[ F[u, v] \]

\[ F \]

Spatial Locations
Stacked to form Vector

\[ X[n_x, n_y] \]
DFT as a Co-ordinate Transform

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

DFT as a Co-ordinate Transform

\[ F = \frac{1}{\sqrt{WH}} S^* X, \quad X = \sqrt{WH} S F \]

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

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

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

\[ X \]
FOURIER TRANSFORM

\[ X \]

\[ |F|^2 \]
FOURIER TRANSFORM

$$X$$

$$|F|^2$$

$$(W-u) \quad 0 \quad u$$

Zero-centered Co-ordinates for frequencies $[u,v]$
FOURIER TRANSFORM

\[ X \quad |F|^2 \quad \angle F \]
FOURIER TRANSFORM

$x$

$|F|^2$

$\angle F$

Reconstruct with only these frequency components
FOURIER TRANSFORM

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
FOURIER TRANSFORM
FOURIER TRANSFORM

A

Magnitude A
Phase B

B

Magnitude B
Phase A
FOURIER TRANSFORM

A

Magnitude A
Phase B

B

Magnitude B
Phase A
FOURIER TRANSFORM

Location of edges / structure, defined by phase more than magnitude.
The FT gave us a different representation for images. Decomposing image into different frequency ‘components’.

What else?
CONVOLUTION THEOREM

Convolution in "matrix" form

Spatial Locations
Stacked to form Vector

$X[n_x, n_y]$
CONVOLUTION THEOREM

Convolution in "matrix" form

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

Spatial Locations
Stacked to form Vector
CONVOLUTION THEOREM

Convolution in "matrix" form

\[ Y[n_x, n_y] \]

Spatial Locations
Stacked to form Vector

\[ X[n_x, n_y] \]

Spatial Locations
Stacked to form Vector

\[ Y[n_x, n_y] = \sum_{n_x'} \sum_{n_y'} k[n_x', n_y'] \ X[n_x - n_x', n_y - n_y'] \]
Convolution in "matrix" form

\[ Y[n_x, n_y] \rightarrow Y = \sum_{n'_x} \sum_{n'_y} k[n'_x, n'_y] \cdot X[n_x - n'_x, n_y - n'_y] \rightarrow X[n_x, n_y] \]

Spatial Locations Stacked to form Vector
Spatial Locations Stacked to form Vector
**CONVOLUTION THEOREM**

Convolution in "matrix" form

\[ Y[n_x, n_y] \begin{bmatrix} \vdots \end{bmatrix} = \begin{bmatrix} \sum_{n'_x} \sum_{n'_y} k[n'_x, n'_y] X[n_x - n'_x, n_y - n'_y] \end{bmatrix} \]

- Mostly 0 (sparse)
- Has \( w_x, h_x \) non-zero entries per row.
- Same set of values, but at different places in each row

Spatial Locations Stacked to form Vector

Spatial Locations Stacked to form Vector
CONVOLUTION THEOREM

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

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

Question:

Let \( Y = A_k X \) correspond to \( Y = X \ast \text{valid} \ k \). 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 \( k \))
CONVOLUTION THEOREM

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

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

\[
\begin{align*}
X[W + n_x, n_y] &= X[n_x, n_y] \\
X[n_x, -n_y] &= X[n_x, H - n_y]
\end{align*}
\]

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

\[ A_k = S D_k S^* \]

- 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!

\[ Y = A_k X = S D_k S^* X \Rightarrow (S^*Y) = D_k(S^*X) \]
**CONVOLUTION THEOREM**

Why does this happen?

- \( X = \sqrt{WH} \sum_{u,v} F[u, v] S_{uv} \)
- \( Y = X \ast k = \sqrt{WH} \sum_{u,v} F[u, v] S_{uv} \ast k \) (by linearity / distributivity)
- \( (S_{uv} \ast k)[n] = \sum_{n'} k[n'] S_{uv}[n - n'] \)
- \( S_{uv}[n - n'], \) 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.
- \( (S_{uv} \ast k)[n_x, n_y] = d_{uv;k} S_{uv}[n_x, n_y], \) where \( d_{uv;k} \) is some complex scalar.

*Sinusoids are eigen-functions of convolution*

\[
Y = X \ast k = \sqrt{WH} \sum_{u,v} F[u, v] S_{uv} \ast k = \sqrt{WH} \sum_{u,v} \left( F[u, v] d_{uv;k} \right) S_{uv}
\]
CONVOLUTION THEOREM

\[ A_k = S D_k S^* \]

- 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.
Kernel has to be the same size as the image.

1. Zero-pad
Kernel has to be the same size as the image.

1. Zero-pad
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)
Kernel has to be the same size as the image.
- From same circular, you can always get 'valid' by cropping.
- To get full / same with zero-padding, pad your original image first.

1. Zero-pad
2. Circularly shift to center at (0,0)
Kernel / Fourier Transform (magnitude) Pairs
Kernel / Fourier Transform (magnitude) Pairs

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

Kernel / Fourier Transform (magnitude) Pairs

Kernel

Magnitude

Kernel

Magnitude

Gaussian Derivatives: Band-pass

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

Kernel / Fourier Transform (magnitude) Pairs

Kernel

Magnitude

Kernel

Magnitude

Gaussian Derivatives: Band-pass

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

For more indepth coverage:
Szeliski Sec 3.4
SCALE & ALIASING

"Resize" Images

(W/2) x (H/2)
"Resize" Images

\[
\text{W x H}
\]

\[(W/2) \times (H/2)\]
"Resize" Images
SCALE & ALIASING

"Resize" Images

[W x H]

(W/2) x (H/2)

[Source: Wikipedia]
SCALE & ALIASING

"Resize" Images

"Aliasing"

[Source: Wikipedia]
Remember, in the two cases $F[u,v]$ is defined with respect to different width and height $W_x$ and $H_x$, 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 $W_x$ and $H_x$, 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.

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

Remember, in the two cases $F[u,v]$ is defined with respect to different width and height $W_x$ and $H_x$, and for different ranges of $(u,v)$.
Make sure there are no high frequencies before sub-sampling!

Low-pass filter, i.e., Smooth Image before sub-sampling.
SCALE & ALIASING

Without Smoothing

With Smoothing
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).
"Resize" Images

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

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

- Need to hallucinate missing information.
- Lots of research (super-resolution).
- Simplest Approach: Nearest neighbor
- Simple Approach: (Bi) Linear Interpolation
"Resize" Images

- Need to hallucinate missing information.
- Lots of research (super-resolution).
- Simplest Approach: Nearest neighbor
- Simple Approach: (Bi) Linear Interpolation
"Resize" Images

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

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

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.
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.
EFFICIENT COMPUTATION

- Convolution, in the most general case, takes $O(n_x n_k)$ time.
  - $n_x = W_x H_x, n_k = W_k H_k$.
- Convolution in the frequency domain:
  - FFT, point-wise multiply, Inverse FFT
  - FFT/IFFT complexity is $O(n_x \log n_x)$ (Most efficient for power of 2 image size)
  - May be worth it for large kernels
  - Or same image convolved with many different kernels
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 \( k[n_x, n_y] = k_x[n_x]k_y[n_y] = k_x \ast_{\text{full}} k_y \).

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 \( W_kH_k \).

- 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!)