Lecture 3: Linear Filters

Professor Fei-Fei Li
Stanford Vision Lab
What we will learn today?

• Images as functions
• Linear systems (filters)
• Convolution and correlation
• Discrete Fourier Transform (DFT)
• Sampling and aliasing

Some background reading:
Forsyth and Ponce, Computer Vision, Chapter 7 & 8
Jae S. Lim, Two-dimensional signal and image processing, Chapter 1, 4, 5
Images as functions

• **An Image** as a function \( f \) from \( \mathbb{R}^2 \) to \( \mathbb{R}^M \):

  • \( f(x, y) \) gives the **intensity** at position \((x, y)\)
  
• Defined over a rectangle, with a finite range:

\[
 f: [a,b] \times [c,d] \rightarrow [0,255]
\]
Images as functions

• **An Image** as a function $f$ from $\mathbb{R}^2$ to $\mathbb{R}^M$:

  • $f(x, y)$ gives the **intensity** at position $(x, y)$
  • Defined over a rectangle, with a finite range:

    $f: [a, b] \times [c, d] \rightarrow [0, 255]$  

  • A color image: $f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$
Images as discrete functions

• Images are usually **digital (discrete)**:
  – **Sample** the 2D space on a regular grid

• Represented as a matrix of integer values

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>79</td>
<td>23</td>
<td>119</td>
<td>120</td>
<td>05</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>9</td>
<td>62</td>
<td>12</td>
<td>78</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>58</td>
<td>197</td>
<td>46</td>
<td>46</td>
<td>0</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>176</td>
<td>135</td>
<td>5</td>
<td>188</td>
<td>191</td>
<td>68</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>26</td>
<td>37</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>0</td>
<td>89</td>
<td>144</td>
<td>147</td>
<td>187</td>
<td>102</td>
<td>62</td>
<td>208</td>
</tr>
<tr>
<td>255</td>
<td>252</td>
<td>0</td>
<td>166</td>
<td>123</td>
<td>62</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>166</td>
<td>63</td>
<td>127</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>99</td>
<td>30</td>
</tr>
</tbody>
</table>
```
Images as discrete functions

Cartesian coordinates

\[ f[n, m] = \begin{bmatrix}
  \vdots & f[-1, 1] & f[0, 1] & f[1, 1] \\
  \vdots & f[-1, 0] & f[0, 0] & f[1, 0] \\
  f[-1, -1] & f[0, -1] & f[1, -1] & \vdots \\
  \vdots & \vdots & \vdots & \vdots 
\end{bmatrix} \]

Notation for discrete functions
Images as discrete functions

Array coordinates

\[
A = \begin{bmatrix}
  a_{11} & \ldots & a_{1M} \\
  \vdots & \ddots & \vdots \\
  a_{N1} & \ldots & a_{NM}
\end{bmatrix}
\]

Matlab notation
Systems and Filters

• Filtering:
  – Form a new image whose pixels are a combination original pixel values

Goals:
- Extract useful information from the images
  • Features (edges, corners, blobs...)

- Modify or enhance image properties:
  • super-resolution; in-painting; de-noising
2D discrete-space systems (filters)

\[ f[n, m] \rightarrow \text{System } S \rightarrow g[n, m] \]

\[ g = S[f], \quad g[n, m] = S\{f[n, m]\} \]

\[ f[n, m] \xrightarrow{S} g[n, m] \]
Filters: Examples

- 2D DS moving average over a $3 \times 3$ window of neighborhood

\[
g[n, m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k, l]
\]

\[
= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n - k, m - l]
\]

\[
(f * h)[m, n] = \frac{1}{9} \sum_{k,l} f[k, l] h[m - k, n - l]
\]
Moving average

\[ F[x, y] \quad G[x, y] \]
Moving average

\[ F[x, y] \quad G[x, y] \]

\[
(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]
\]
Moving average

\[ F[x, y] \]

\[ G[x, y] \]

\[(f \ast g)[m, n] = \sum_{k,l} f[k, l] g[m-k, n-l]\]
Moving average

\[ F[x, y] \]

\[ G[x, y] \]

\[
(f \ast g)[m, n] = \sum_{k,l} f[k, l] g[m-k, n-l]
\]
Moving average

\[ F[x, y] \quad \text{and} \quad G[x, y] \]

\[(f \ast g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]\]
Moving average

\[ F[x, y] \quad G[x, y] \]

\[
(f \ast g)[m, n] = \sum_{k,l} f[k, l] g[m-k, n-l]
\]

Source: S. Seitz
Moving average

In summary:

• Replaces each pixel with an average of its neighborhood.

• Achieve smoothing effect (remove sharp features)

\[
\frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]
Moving average
Filters: Examples

• Image segmentation based on a simple threshold:

\[ g[n, m] = \begin{cases} 
1, & f[n, m] > 10 \\
0, & \text{otherwise.} 
\end{cases} \]
Classification of systems

- Amplitude properties
  - Linearity
  - Stability
  - Invertibility

- Spatial properties
  - Causality
  - Separability
  - Memory
    - Shift invariance
    - Rotation invariance
Shift-invariance

• If \( f[n, m] \xrightarrow{s} g[n, m] \) then

\[
\begin{align*}
    f[n - n_0, m - m_0] & \xrightarrow{s} g[n - n_0, m - m_0] \\
\end{align*}
\]

for every input image \( f[n,m] \) and shifts \( n_0, m_0 \)

• Is the moving average shift invariant a system?
Is the moving average system shift invariant?

\[ F[x, y] \quad G[x, y] \]
Is the moving average system is shift invariant?

\[
f[n, m] \xrightarrow{S} g[n, m] = \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n - k, m - l] \\
\]

\[
f[n - n_0, m - m_0] \\
\]

\[
\xrightarrow{S} g[n, m] = \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n - k, m - l] \\
\]

\[
= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[(n - n_0) - k, (m - m_0) - l] \\
\]

\[
= g[n - n_0, m - m_0] \quad \text{Yes!} 
\]
Linear Systems (filters)

\[ f(x, y) \rightarrow S \rightarrow g(x, y) \]

- Linear filtering:
  - Form a new image whose pixels are a weighted sum of original pixel values
  - Use the same set of weights at each point

- \( S \) is a linear system (function) iff it satisfies

\[
S[\alpha f_1 + \beta f_2] = \alpha S[f_1] + \beta S[f_2]
\]

superposition property
Linear Systems (filters)

\[ f(x, y) \rightarrow S \rightarrow g(x, y) \]

• Is the moving average a system linear?

• Is thresholding a system linear?
  – \( f_1[n,m] + f_2[n,m] > T \)
  – \( f_1[n,m] < T \)
  – \( f_2[n,m] < T \quad \text{No!} \)
LSI (linear *shift invariant*) systems

Impulse response

\[
\delta_2[n, m] \rightarrow \mathcal{S} \rightarrow h[n, m]
\]

\[
\delta_2[n - k, m - l] \rightarrow \mathcal{S} (\text{SI}) \rightarrow h[n - k, m - l]
\]
LSI (linear shift invariant) systems

Example: impulse response of the 3 by 3 moving average filter:

\[
\begin{align*}
    h[n, m] &= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} \delta_2[n - k, m - l] \\
    &= \begin{bmatrix}
        1/9 & 1/9 & 1/9 \\
        1/9 & 1/9 & 1/9 \\
        1/9 & 1/9 & 1/9 \\
    \end{bmatrix}
\end{align*}
\]
An LSI system is completely specified by its impulse response.

\[
 f[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \delta_2[n - k, m - l] 
\]

\[
 \rightarrow [\mathcal{S} \text{ LSI}] \rightarrow \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l] 
\]

\[
 = f[n, m] \ast h[n, m] 
\]

superposition

Discrete convolution
Discrete convolution

- Fold $h[n,m]$ about origin to form $h[-k,-l]$
- Shift the folded results by $n,m$ to form $h[n - k,m - l]$
- Multiply $h[n - k,m - l]$ by $f[k, l]$
- Sum over all $k,l$
- Repeat for every $n,m$
Discrete convolution

• Fold $h[n,m]$ about origin to form $h[-k,-l]$
• Shift the folded results by $n,m$ to form $h[n-k,m-l]$
• Multiply $h[n-k,m-l]$ by $f[k,l]$
• Sum over all $k,l$
• Repeat for every $n,m$
Discrete convolution

- Fold $h[n,m]$ about origin to form $h[-k,-l]$
- Shift the folded results by $n,m$ to form $h[n-k,m-l]$
- Multiply $h[n-k,m-l]$ by $f[k,l]$
- Sum over all $k,l$
- Repeat for every $n,m$

\[
\sum (f[k,l] \times h[n-k,m-l])
\]

f[k,l] x h[n-k,m-l]
Convolution in 2D - examples

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

= ?

Courtesy of D Lowe
Convolution in 2D - examples

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

Filtered (no change)
Convolution in 2D - examples

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array}
\]

= ?
Convolution in 2D - examples

Original

= 

Shifted left
By 1 pixel
Convolution in 2D - examples

Original

\[
\begin{array}{ccc}
& 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[\frac{1}{9} = ?\]
Convolution in 2D - examples

Original

Blur (with a box filter)
Convolution in 2D - examples

Original

(Note that filter sums to 1)
• What does blurring take away?

original - smoothed (5x5) = detail

• Let’s add it back:

original + detail = sharpened
Convolution in 2D – Sharpening filter

Sharpening filter: Accentuates differences with local average
Convolution properties

• Commutative property:

\[ f ** h = h ** f \]

• Associative property:

\[ (f ** h_1) ** h_2 = f ** (h_1 ** h_2) \]

• Distributive property:

\[ f ** (h_1 + h_2) = (f ** h_1) + (f ** h_2) \]

The order doesn’t matter! \[ h_1 ** h_2 = h_2 ** h_1 \]
Convolution properties

• Shift property:

\[ f[n, m] \star\star \delta_2[n - n_0, m - m_0] = f[n - n_0, m - m_0] \]

• Shift-invariance:

\[ g[n, m] = f[n, m] \star\star h[n, m] \]

\[ \implies f[n - l_1, m - l_1] \star\star h[n - l_2, m - l_2] \]

\[ = g[n - l_1 - l_2, m - l_1 - l_2] \]
Image support and edge effect

• A computer will only convolve finite support signals.
  • That is: images that are zero for n,m outside some rectangular region

• MATLAB’s conv2 performs 2D DS convolution of finite-support signals.

\[
N_1 \times M_1 \ast N_2 \times M_2 = (N_1 + N_2 - 1) \times (M_1 + M_2 - 1)
\]
Image support and edge effect

• A computer will only convolve finite support signals.
• What happens at the edge?

- zero “padding”
- edge value replication
- mirror extension
- more (beyond the scope of this class)

-> Matlab conv2 uses zero-padding
Cross correlation

Cross correlation of two 2D signals $f[n,m]$ and $g[n,m]$

$$r_{fg}[k, l] \triangleq \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n, m] g^*[n - k, m - l]$$

$$= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n + k, m + l] g^*[n, m], \quad k, l \in \mathbb{Z}.$$ 

(k, l) is called the lag

• Equivalent to a convolution without the flip

$$r_{fg}[n, m] = f[n, m] \ast\ast g^*[-n, -m]$$
Cross correlation – example

MATLAB’s `xcorr2`

`g = f + noise`

`g > 0.5`

`r > 0.5`

Courtesy of J. Fessler
Cross correlation – example

Left

Right

scanline

Norm. corr

Fei-Fei Li
Convolution vs. Correlation

• A **convolution** is an integral that expresses the amount of overlap of one function as it is shifted over another function.
  – convolution is a filtering operation

• **Correlation** compares the *similarity of two sets of data*. Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best.
  – correlation is a measure of relatedness of two signals
2D Discrete-Signal Fourier Transform (DSFT)

2D discrete-signal Fourier transform (DSFT) of a 2D discrete-space signal $g[n,m]$:

$$G(\omega_x, \omega_y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} g[n, m] e^{-i(\omega_x n + \omega_y m)}$$

Inverse 2D DSFT:

$$g[n, m] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G(\omega_x, \omega_y) e^{i(\omega_x n + \omega_y m)} d\omega_x d\omega_y$$
Jean Baptiste Joseph Fourier

Fourier’s Transform: log(Magnitude)

Fourier’s Transform: Phase

DSFT

\[ e^{i \angle G(\omega_X, \omega_Y)} \]

\[ |G(\omega_X, \omega_Y)| \]

Fei-Fei Li
DSFT – properties

Shift:

\[ g[n - n_0, m - m_0] \leftrightarrow G(\omega_x, \omega_y) e^{-i(\omega_x n_0 + \omega_y m_0)} \]

Convolution:

\[ f[n, m] \ast h[n, m] \leftrightarrow F(\omega_x, \omega_y) H(\omega_x, \omega_y) \]

Delta function:

\[ \delta_2[n, m] \leftrightarrow 1 \]

\[ \delta_2[n - n_0, m - m_0] \leftrightarrow e^{-i(\omega_x n_0 + \omega_y m_0)} \]
Example: DSFT of moving average filter

\[ g[n, m] = \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n - k, m - l] \]

\[ (f * h)[m, n] = \frac{1}{9} \sum_{k,l} f[k, l] h[m - k, n - l] \]

\[ F(\omega_X, \omega_Y) \cdot H(\omega_X, \omega_Y) = \frac{1}{9} \sum_{n=-1}^{1} \sum_{m=-1}^{1} e^{-\omega_X n} e^{-\omega_Y m} \]

\[ = \frac{1}{9}[1 + 2 \cos \omega_X][1 + 2 \cos \omega_Y] \]
<table>
<thead>
<tr>
<th>Function</th>
<th>Fourier transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g(x, y)$</td>
<td>$\int \int g(x, y)e^{-i2\pi(ux+vy)}dxdy$</td>
</tr>
<tr>
<td>$\int \int \mathcal{F}(g(x, y))(u, v)e^{i2\pi(ux+vy)}dudv$</td>
<td>$\mathcal{F}(g(x, y))(u, v)$</td>
</tr>
<tr>
<td>$\delta(x, y)$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\frac{\partial f}{\partial x}(x, y)$</td>
<td>$u\mathcal{F}(f)(u, v)$</td>
</tr>
<tr>
<td>$0.5\delta(x + a, y) + 0.5\delta(x - a, y)$</td>
<td>$\cos 2\pi au$</td>
</tr>
<tr>
<td>$e^{-\pi(x^2+y^2)}$</td>
<td>$e^{-\pi(u^2+v^2)}$</td>
</tr>
<tr>
<td>box$_1(x, y)$</td>
<td>$\frac{\sin u \sin v}{u \frac{v}{u}}$</td>
</tr>
<tr>
<td>$f(ax, by)$</td>
<td>$\mathcal{F}(f)(u/a, v/b)$</td>
</tr>
<tr>
<td>$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \delta(x - i, y - j)$</td>
<td>$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \delta(u - i, v - j)$</td>
</tr>
<tr>
<td>$(f \ast g)(x, y)$</td>
<td>$\mathcal{F}(f) \mathcal{F}(g)(u, v)$</td>
</tr>
<tr>
<td>$f(x - a, y - b)$</td>
<td>$e^{-i2\pi(au+bv)}\mathcal{F}(f)$</td>
</tr>
<tr>
<td>$f(x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$</td>
<td>$\mathcal{F}(f)(u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta)$</td>
</tr>
</tbody>
</table>
Why is DFT important?

• Perform efficient linear convolution as product of DFTs
• Each DFT can be implemented using the FFT (Fast Fourier Transform) [see appendix for details]
Sampling

Throw away every other row and column to create a 1/2 size image
Sampling

• Down-sampling operation:

(trivial form of image compression)

\[ g[n, m] = f[2n, 2m] = \begin{bmatrix}
\vdots & \vdots & \vdots \\
\cdots & f[-2, 2] & f[0, 2] & f[2, 2] \\
\cdots & f[-2, 0] & f[0, 0] & f[2, 0] \\
\vdots & \vdots & \vdots 
\end{bmatrix} \]
Why is a multi-scale representation useful?

• Find template matches at all scales
  – e.g., when finding hands or faces, we don’t know what size they will be in a particular image
  – Template size is constant, but image size changes

• Efficient search for correspondence
  – look at coarse scales, then refine with finer scales

• Examining all levels of detail
  – Find edges with different amounts of blur
  – Find textures with different spatial frequencies (levels of detail)

Slide credit: David Lowe (UBC)
Aliasing

Disintegrating textures
Sampling Theorem (Nyquist)

• When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{\text{max}}$
• $f_{\text{max}} = \text{max frequency of the input signal.}$
N = sampling period
T = periodicity of the replicas of the DFT (g)

T \sim 1/N
gsampled \[n,m\]

Large sampling period \(N\)

\[\omega_x\]

\[\omega_y\]

\[G_d(\omega_x, \omega_y)\]

- \(N = \text{sampling period}\)
- \(T = \text{periodicity of the replicas of the DFT (g)}\)

\(T \sim 1/N\)
Anti-aliasing

Solutions:
• Sample more often

• Get rid of all frequencies that are greater than half the new sampling frequency
  – Will lose information - but it’s better than aliasing
  – Apply a smoothing filter to remove high frequencies
Anti-aliasing

Apply a smoothing filter to remove high frequencies:
Sampling algorithm

Algorithm 7.1: Sub-sampling an Image by a Factor of Two

Apply a low-pass filter to the original image
(a Gaussian with a $\sigma$ of between one
and two pixels is usually an acceptable choice).
Create a new image whose dimensions on edge are half
those of the old image
Set the value of the $i, j$’th pixel of the new image to the value
of the $2i, 2j$’th pixel of the filtered image
Resampling with Prior Smoothing

• Note: We cannot recover the high frequencies, but we can avoid artifacts by smoothing before resampling.

Image Source: Forsyth & Ponce
The Gaussian Pyramid

\[ G_4 = (G_3 \ast \text{gaussian}) \downarrow 2 \]
\[ G_3 = (G_2 \ast \text{gaussian}) \downarrow 2 \]
\[ G_2 = (G_1 \ast \text{gaussian}) \downarrow 2 \]
\[ G_1 = (G_0 \ast \text{gaussian}) \downarrow 2 \]
\[ G_0 = \text{Image} \]
Gaussian Pyramid – Stored Information

All the extra levels add very little overhead for memory or computation!

Source: Irani & Basri
Summary: Gaussian Pyramid

- **Construction:** create each level from previous one
  - Smooth and sample

- **Smooth with Gaussians, in part because**
  - a Gaussian*Gaussian = another Gaussian
  - \( G(\sigma_1) \ast G(\sigma_2) = G(\sqrt{\sigma_1^2 + \sigma_2^2}) \)

- **Gaussians are low-pass filters, so the representation is redundant once smoothing has been performed.**
  - \( \Rightarrow \) There is no need to store smoothed images at the full original resolution.
Application: Vision system for TV remote control
- uses template matching

What we have learned today?

• Images as functions
• Linear systems (filters)
• Convolution and correlation
• Discrete Fourier Transform (DFT)
• Sampling and aliasing
Appendix
Convergence

If $g$ absolutely summable:

$$\sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} |g[n, m]| < \infty$$

then

$$\lim_{N \to \infty} \sum_{n=-N}^{N} \sum_{m=-N}^{N} g[n, m] e^{i(\omega_X n + \omega_Y m)} = G(\omega_X, \omega_Y)$$

If $g$ is square summable (energy signal):

$$E_g \triangleq \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} |g[n, m]|^2 < \infty$$

$$\int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G_N(\omega_X, \omega_Y) - G(\omega_X, \omega_Y)^2 \, d\omega_X \, d\omega_Y \to 0$$
Fast Fourier transforms (FFT)

• Brute-force evaluation of the 2D DFT would require $O((NM)^2)$ flops

$$X[k, l] \triangleq$$

$$= \left\{ \begin{array}{ll}
\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x[n, m] e^{-i2\pi(kn/N+lm/M)}, \\
0,
\end{array} \right.$$

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

otherwise.

• DFT is a **separable operation**

  ➔ we can reduce greatly the computation
Fast Fourier transforms (FFT)

• DFT is a **separable operation**:

\[
X[k, l] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x[n, m] e^{-i2\pi(kn/N + lm/M)}
\]

\[
= \sum_{n=0}^{N-1} e^{-i2\pi kn/N} \left[ \sum_{m=0}^{M-1} x[n, m] e^{-i2\pi lm/M} \right]
\]

• Apply the 1D DFT to each column of the image, and then apply the 1D DFT to each row of the result.

• Use the **fast Fourier transform (1-D FFT)** for these 1D DFTs!
Fast Fourier transforms (FFT)

• FFT computational efficiency:

  • inner set of 1D FFTs: \( N \mathcal{O}(M \log M) \)
  • outer set of 1D FFTs: \( M \mathcal{O}(N \log N) \)

  • **Total:** \( \mathcal{O}(MN \log MN) \) flops

• A critical property of FFT is that \( N = 2^k \) with \( k \) = integer

If \( x = 512 \times 512 \) image \( \Rightarrow \)
saving is a factor of 15000 relative to the brute-force 2D DFT!

Matlab: \( \text{fft2} = \text{fft}(	ext{fft}(x).') \) \( \)
FFT & Efficiency

• **General goal:** perform efficient linear convolution
• Perform convolution as product of DFTs
• **Pros:** DFT can be implemented using the FFT (fast fourier transform)
  • FFT is very efficient (fast!)

• **Cons:** DFT perform circular convolution
  • Compensate the wrap-around effect

• **Cons:** Online-memory storage
  • Use the overlap-add method or overlap-save method
FFT & Efficiency

• Suppose we wish to convolve a $256 \times 256$ image with a $17 \times 17$ filter.

• The result will be $272 \times 272$.

• The smallest prime factors of 272 is 2.

• So one could pad to a $512 \times 512$ image
  • Note: only 28% of the final image would be the part we care about - the rest would be zero in exact arithmetic.

• Handling to a $512 \times 512$ image requires much memory

→ Use overlap-add method