**STORY SO FAR**

- **Convolutions**
  - Linear operations on images
  - Each pixel of output is linear weighted sum of neighborhood in input
  - Same weights for all neighborhoods
  - Can be "diagonalized" in the Fourier Domain

But so far, output image size is (approximately) equal to input image size

---

**SCALE & ALIASING**

W x H

(W/2) x (H/2)

"Resize" Images

---

"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 $W_w$ and $H_h$, 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.

"Resize" Images

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

$$Y[n] = X[\text{round}(n/2)]$$
Convolution, in the most general case, takes $O(n_x n_y)$ time.

- $n_x = W_x H_x, n_y = W_y H_y$

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 $k[n_x, n_y] = k_x[n_x]k_y[n_y] = k_x \ast_w k_y$.
  
  This is by interpreting $k_x$ and $k_y$ as having size $W_x \times 1$ and $1 \times H_y$.

- So $X \ast k = X \ast (k_x \ast k_y) = (X \ast k_x) \ast k_y$. This takes $W_x + H_y$ operations instead of $W_x H_y$.

- Often if a kernel itself isn't separable, it can be sometimes expressed as a sum of separable kernels.
- E.g., Unsharp Mask: $(1 + \alpha)\delta - \alpha G_x$ (don't combine!)
- Could also try to do this automatically using SVD.

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

- Sometimes can decompose into convolution with sparse kernels.
- Many implementations of `convolve2d` won’t make use of sparsity.
  - But you can write your own.

Smooth and Sub-sample

- Don’t smooth and sub-sample!
- For sub-sampling by two, you’re computing 4x as many smooth filter outputs as you need to.

**numpy Specifics**

1. In general, prefer algorithms that have lower total number of multiplies / adds.
2. Try to use `scipy.signal.convolve2d` (subject to rule 1). It is optimized for cache reads, parallel execution, etc.
   (I import it often as `from scipy.signal import convolve2d as conv2`)
3. Similarly, avoid for loops and use element-wise operations on large arrays, matrix multiplies (np.matmul / np.dot), etc.

- Some of these things are faster in python because a single large operation runs natively instead of returning to the compiler. But they’re also faster because these operations are often 'atomics' in lower-level libraries too (BLAS), and have been highly optimized for modern hardware.
- Thinking about designing your algorithm in terms of these atomic operations is useful beyond python.
- Some points in problem sets allocated for efficient code.

**MULTI-SCALE REPRESENTATIONS**

Fourier Transform useful for:

- Diagonalizing Convolution: $A_k = S D_k S^*$
- Concentrating energy (high manitudes) in fewer coefficients

But is the FT interpretable? Does it tell us something about the image?
MULTI-SCALE REPRESENTATIONS

- $F[u, v]$ is intuitively average variation in image at that frequency.
- But averaged across the entire image.
- This isn't useful because images aren't "stationary"
- Different parts of the image, "have different frequencies".

FT decomposition of different levels (coarse/fine) of variation: but without sense of spatial location.
Multi-scale representations aim to address this.

MULTI-SCALE REPRESENTATIONS

Gaussian Pyramid

Useful for analyzing image at multiple scales.
E.g., apply the same method (edge detection / CNN) at multiple levels of the pyramid.

MULTI-SCALE REPRESENTATIONS

Laplacian Pyramid

Keep Gaussian at Coarsest Level.
**MULTI-SCALE REPRESENTATIONS**

**Laplacian Pyramid**

No orientation selectivity: See Steerable Pyramids
http://www.cns.nyu.edu/~eero/steerpyr/

**MULTI-SCALE REPRESENTATIONS**

**Wavelet Pyramid**

Gaussian & Laplacian pyramids are good for analysis, but not really a representation. Not easy / possible to go from pyramid coefficients back to original image (like Fourier Transform).

Consider a 2x2 Pixel Block

\[
\begin{bmatrix}
  a & c \\
  b & d
\end{bmatrix}
\]

\[
L = \frac{a + b + c + d}{2}, \quad H_2 = \frac{e + d - a - b}{2}
\]

\[
H_1 = \frac{b + d - a - c}{2}, \quad H_3 = \frac{a + d - b - c}{2}
\]

\[
\begin{bmatrix}
  L \\
  H_1 \\
  H_2 \\
  H_3
\end{bmatrix} = \frac{1}{2} \begin{bmatrix}
  1 & 1 & 1 & 1 \\
  -1 & 1 & -1 & 1 \\
  -1 & -1 & 1 & 1 \\
  1 & -1 & -1 & 1
\end{bmatrix} \begin{bmatrix}
  a \\
  b \\
  c \\
  d
\end{bmatrix}
\]

Unitary Matrix / Co-ordinate transform
**MULTI-SCALE REPRESENTATIONS**

**Wavelet Pyramid**

![Wavelet Pyramid Diagram](image)

\[ L = \frac{a + b + c + d}{2} \]

\[ H_L = \frac{b + d - a - c}{2} \]

\[ H_L = \frac{a + d - b - c}{2} \]

**Applications:** Analysis, image modeling & restoration, compression