CONVENTION

RECAP

- An image $X$ is an array* of intensities.
- $X[n]$ or $X[n_x, n_y]$ refers to intensities for a particular pixel at location $n$ or $[n_x, n_y]$.
  - Single index denotes $n = [n_x, n_y]^T$ is a vector of two integers.
- Each $X[n]$ is a scalar for a grayscale image, or a 3-vector for an RGB color image.
  (Unless otherwise specified, vector implies column vector)

$$X[n] = [I_x, I_y, I_z]$$

*Clarification: numpy convention is H x W x C: (vertical, horizontal, channels) or H x W.

Do not think of single-channel images themselves as matrices!
It makes no sense to “matrix multiply” a 80x60 pixel image with a 60x20 pixel image.

CONVENTION: LINEAR OPERATIONS

- But sometimes, we want to interpret operations as linear on all intensities / intensity vectors in an image.
- Stack all pixel locations, in some pre-determined order, as rows. Represent $X$ as:
  - $(HW) \times 3$ matrix: color images
  - $(HW) \times 1$ vector: grayscale images.

$$Y[n] = C \cdot X[n] \Rightarrow Y = X \cdot C^T$$

### OFFICE HOURS

- Jarett Gross: Mon 5:40pm-6:30pm @ TBD
- Ayan Chakrabarti: Wed 9:30am-10:30am @ Jolley 205
- Abby Stylianou*: Fri 10:00am-11:00am @ TBD

Mon/Fri locations will be decided in a day or two.

* Some of the Friday slots will be allocated as recitation sections (one for each problem set).
  Dates will be posted in advance.

```python
# Begin with X as (H,W,3) array
Xflt = np.reshape(X, (-1, 3))  # Flatten X to a (HW, 3) matrix
Yflt = np.matmul(Xflt, C.T)    # Post-multiply by C
Y = np.reshape(Yflt, X.shape)  # Turn Y back to an image array
```
CONVOLUTION

Notation: \( Y = X \ast k \)

\[
Y[n] = \sum_{n'} k[n'] \ X[n - n']
\]

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

- Double summation over the support / size of the kernel \( k \)
- We assume \( k[n] \in \mathbb{R} \) is scalar valued.
  - If \( X[n] \) is scalar, so is \( Y[n] \).
  - If \( X \) is a color image, each channel convolved with \( k \) independently.

To go from \( m \) to \( n \) channels in a "conv layer": \( k[n] \in \mathbb{R}^{m \times m} \) is matrix valued, and \( k[n'] \ X[n - n'] \) is a matrix-vector product.

\[
Y[n] = \sum_{n'} k[n'] \ X[n - n']
\]

This assumes a 0 centered kernel

CONVOLUTION

We pass 2D arrays to the convolve functions, and get a 2D array out. Let’s assume top left index is \( (0,0) \) for all.

Let \( W_x, W_k \) and \( W_y \) denote the widths of \( X, k \), and \( Y \); and \( H_x, H_k \) and \( H_y \) the heights.

The 2D convolution function in most libraries provide 3 options: Valid, Full, and Same.

\[
Y[n] = \sum_{n'} k[n'] \ X[n - n']
\]

Valid: Subset of values of \( Y[n] \) for which EVERY \( X[n - n'] \) is defined.

\[
Y[n] = \sum_{n'} k[n'] \ X[n - n']
\]
CONVOLUTION

CONVOLUTION: PROPERTIES

Let $X \ast_w k$, $X \ast_v k$, and $X \ast_{\text{same}} k$ denote full, valid, and same convolution (with zero padding for full and same).

- **Linear / Distributive**: For scalars $\alpha, \beta$;
  - If $Y = X \ast k$, then: $X \ast (\alpha k) = (\alpha X) \ast k = \alpha Y$
  - If $Y_1 = X \ast k_1$ and $Y_2 = X \ast k_2$ (same size): $X \ast (\alpha k_1 + \beta k_2) = \alpha Y_1 + \beta Y_2$

- **Associative**
  - $(X \ast_w k_1) \ast_w k_2 = X \ast_w (k_1 \ast_w k_2)$
  - $(X \ast_v k_1) \ast_v k_2 = X \ast_v (k_1 \ast_v k_2)$
  - $(X \ast_{\text{same}} k_1) \ast_{\text{same}} k_2 \neq X \ast_{\text{same}} (k_1 \ast_{\text{same}} k_2)$

- **Commutative**:
  - $k_1 \ast_w k_2 = k_2 \ast_w k_1$
  - $(X \ast_w k_1) \ast_w k_2 = (X \ast_w k_2) \ast_w k_1$
  - $(X \ast_v k_1) \ast_v k_2 = (X \ast_v k_2) \ast_v k_1$
  - $(X \ast_{\text{same}} k_1) \ast_{\text{same}} k_2 \neq (X \ast_{\text{same}} k_2) \ast_{\text{same}} k_1$

Full: Subset of values of $Y[n]$ for which ANY $X[n - n']$ is defined.

$$Y[n] = \sum_{n'} k[n'] \ X[n - n']$$

Padding

What do we use for the missing values of $X[n]$?

- Zero (Often Default)
- Some other constant
- Reflect / Symmetric (across boundary)
- Circular (wrap around)
- Replicate

...
CONVOLUTION

\( X[n] \)

\( Y[n] \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\( k[n] \)

\( \text{Kernel = Impulse Response} \)

\( Y'[n] \)

\( k[n] \)

\( k[n] = 1 / 25 \)

CONVOLUTION

\( X[n] \)

\( Y'[n] \)

\( X[n] \)

\( Y'[n] \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\( k[n] \)

\( \text{Same with zero padding} \)

\( k[n] \)

\( \text{Same with zero padding} \)
CONVOLUTION

Gaussian Kernels

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

\[ \sum_{n_x, n_y} G_{\sigma}[n_x, n_y] = 1 \]

\( n_x, n_y = [-S, -(S - 1), \ldots, -1, 0, 1, \ldots, (S - 1), S] \)

\( \sigma = 1 \)

\( \sigma = 2 \)

\( \sigma = 3 \)

\( \sigma = 4 \)
CONVOLUTION

Gaussian Kernels

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

\[ \sum_{n_x, n_y} G_{\sigma}[n_x, n_y] = 1 \quad n_x, n_y = [-S, -S, \ldots, 0, \ldots, S-1, S] \]

CONVOLUTION

Unsharp Masking

\[ Y = (1 + \alpha)(X - \alpha(X \ast G_{\sigma})) = X \ast ((1 + \alpha) \delta - \alpha G_{\sigma}) \]

\[ \sigma = 4 \quad \alpha = 1 \]

CONVOLUTION

Unsharp Masking

\[ Y = (1 + \alpha)(X - \alpha(X \ast G_{\sigma})) = X \ast ((1 + \alpha) \delta - \alpha G_{\sigma}) \]

\[ \sigma = 2 \quad \alpha = 5 \]

CONVOLUTION

Unsharp Masking

\[ Y = (1 + \alpha)(X - \alpha(X \ast G_{\sigma})) = X \ast ((1 + \alpha) \delta - \alpha G_{\sigma}) \]
CONVOLUTION

\( X[n] \quad Y[n] \quad \sigma = 2 \quad \alpha = 10 \)

Unsharp Masking

\[ Y = (1 + \alpha)X - \alpha(X * G_\sigma) = X * ((1 + \alpha)\delta - \alpha G_\sigma) \]

APPLICATION: EDGE DETECTION

What is an edge?

Different answers from different people!

Depth boundary / Material Boundary / Object Boundary?

Edge (not boundary): Location where image intensity is changing rapidly in some direction.

Directional Derivative

APPLICATION: EDGE DETECTION

Finite Difference Approximation

\[ \frac{\partial}{\partial n_x} X[n_x, n_y] \approx X[n_x + 1, n_y] - X[n_x - 1, n_y] \]

Derivative is a linear spatially invariant operation: Convolution

\[ X * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \]

Smoothed in y direction

"Sobel" Operator

\[ X * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \]

Y Derivative

\[ X * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]

APPLICATION: EDGE DETECTION

\[ X * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \]

Derivatives have been scaled so that gray (0.3) corresponds to 0. Bright to positive derivative values, dark to negative.
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_x = \partial_x \ast (G_\sigma \ast X) = (\partial_x \ast G_\sigma) \ast X = G_{x;\sigma} \ast X \]

\[ G_x = \frac{-x}{2\pi\sigma^4} \exp\left(\frac{-x^2 + y^2}{2\sigma^2}\right) \]

\[ G_y = \frac{-y}{2\pi\sigma^4} \exp\left(\frac{-x^2 + y^2}{2\sigma^2}\right) \]

Derivative of Gaussian (DoG) Filters

\[ G_\theta = \frac{-(x \cos \theta + y \sin \theta)}{2\pi\sigma^4} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \]

\[ I_{\theta} = I_x \cos \theta + I_y \sin \theta \]

Just need to convolve twice. Gives us an expression for derivative along every direction.

APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_{\theta}[n] = I_x[n] \cos \theta + I_y[n] \sin \theta \]

\[ H[n] = \sqrt{I_x^2[n] + I_y^2[n]} = \max_\theta I_{\theta}[n] \]

\[ \Theta[n] = \text{atan2}(I_y[n], I_x[n]) = \arg \max_\theta I_{\theta}[n] \]

Gives us gradient magnitude and direction.

Often applied even to filters that aren’t “steerable” like DoG.

APPLICATION: EDGE DETECTION

\[ I_x \quad I_y \quad I_{45^\circ} \]
Extensions

- Non-maxima Suppression: Keep an edge pixel only if its magnitude is higher than its neighbors along the direction of the derivative.

Declare edge if $a$ above threshold and:
- $a > b$ and $a > c$ if $\theta = 0$
- $a > f$ and $a > j$ if $\theta = 90$
- $a > e$ and $a > k$ if $\theta = 45$
- ....

- Canny: Keep a lower magnitude edge pixel if it has a higher edge magnitude neighbor.
  Two thresholds (hysteresis)

- Second derivative filters.

See Szeliski Section 4.2
Hough Transform
- Consider all possible lines (on a 2D plane)
- This is a two-dimensional search space, could parameterize it in different ways.

\[ r = x \cos \theta + y \sin \theta \]
\[ \theta \in [-\pi/2, \pi/2] \]
\[ r \in [-r_{\text{max}}, r_{\text{max}}] \]

Missed detections
Clutter
Occlusions

Pool them together to detect scene structure: E.g., Lines

Edges are isolated per-pixel labels

Discretize this space into a bunch of buckets.
**OTHER NEIGHBORHOOD OPERATIONS**

**Median Filter / Order Statistics**

\[ Y[n] = \text{Median}\{X[n - n']\}_{N[n'] = 1} \]

- Neighborhood function \( N[n'] \in \{0, 1\} \)
- Often better at removing outliers than convolution.

- Other ops: \( Y[n] = \max / \min \{X[n - n']\}_{N[n'] > 0} \)

**Morphological Operations**

- Conducted on binary images \( X[n] \in \{0, 1\} \)
- Erosion: \( Y[n] = \text{AND} \{X[n - n']\}_{N[n'] = 1} \) (1 if all neighbors 1)
- Dilation: \( Y[n] = \text{OR} \{X[n - n']\}_{N[n'] = 1} \) (1 if any neighbor 1)
- Opening: Erosion followed by Dilation
- Closing: Dilation followed by Erosion

See Szeliski Sec 3.3.2
NEXT TIME

- Bilateral Filtering
- Fourier Transforms
- Making convolutions efficient
- Sampling and Scale
- Image Representations