Out of 79 people enrolled and waitlisted:

- 64 (-15) have submitted their public key.
- 59 (-5) have cloned the PSET 0 repo.
- 47 (-12) have finished and submitted problem set 0.

- Submit public key and make sure you can clone the repo ASAP.
  - Note that in the command: git clone cse559@euclid.seas.wustl.edu:wustl.key/psetN
    - Replace wustl.key with your WUSTL key username
    - Replace psetN with pset0, pset1, etc.
- Finish and submit pset 0 by end of Thursday.

- A couple of people haven't completed sign up form. Do that TODAY.
Comments about Submitted problem sets

- **COMPLETE THE INFORMATION SECTION**
  - Do this for every problem set
  - Tell us how much time the problem set took
  - Let us know who you had discussions with, what external resources you used.
  - If you used none, say that. But don't leave it blank, at the default template.
  - **Read the collaboration and late policies**
- Replace the placeholders for your name / WUSTL key in the tex file header.
- Do a "git pull; git log" after every push to read confirmation commit message.
PSET 1 is out early

- Available to clone. Go ahead take a look.
- We haven't covered all the material yet.
  - By today's class, we should cover everything you need for the first 3-4 problems.
- Not a bad idea to use the extra time to get started early.
- Lecture slides are being posted after class to the course website
  - First two lectures already up
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)

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

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.
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:

- $(H \times W)$ × 3 matrix: color images
- $(H \times W)$ × 1 vector: grayscale images.

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

# Begin with $X$ as $(H,W,3)$ array
Xflt = np.reshape(X, (-1, 3))  # Flatten X to a $(H \cdot W, 3)$ 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: \( Y = X * k \)

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

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

- 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": \( k[n] \in \mathbb{R}^{n \times m} \) is matrix valued, and \( k[n'] X[n - n'] \) is a matrix-vector product.
CONVOLUTION

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

This assumes a 0 centered kernel.
CONVOLUTION

What is the range / size of $Y$?

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

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

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

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

\[ W_y = W_x - W_k + 1 \]
\[ H_y = H_x - H_k + 1 \]
CONVOLUTION

\[ W_y = W_x + W_k - 1 \]
\[ H_y = H_x + H_k - 1 \]

Full: Subet of values of \( Y[n] \) for which ANY \( X[n - n'] \)
is defined.

\[ Y[n] = \sum_{n'} k[n'] \cdot X[n - n'] \]
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
...
Same: Center Crop of Full output, that is same size as X.

For odd sized kernels, corresponds to treating center of kernel as (0,0).

Same does what we "expect", but you should understand the padding and cropping involved. When kernel size isn't odd, which crop is taken often depends on the library.
Let $X \ast_{\text{full}} k, X \ast_{\text{val}} k,$ 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 $Y_2 = X \ast k_2$, $(k_1, k_2$ same size): $X \ast (\alpha k_1 + \beta k_2) = \alpha Y_1 + \beta Y_2$
  - If $Y_1 = X_1 \ast k$ and $Y_2 = X_2 \ast k$, $(X_1, X_2$ same size): $(\alpha X_1 + \beta X_2) \ast k = \alpha Y_1 + \beta Y_2$

\[
X \ast (\alpha k_1 + \beta k_2) [n] = \sum_{n'} (\alpha k_1 [n'] + \beta k_2 [n'])X[n - n'] \\
= \alpha \sum_{n'} k_1 [n']X[n - n'] + \beta \sum_{n'} k_2 [n']X[n - n']
\]
CONVOLUTION: PROPERTIES

Let \( X \ast_{\text{full}} k, X \ast_{\text{val}} k, \) 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 \( Y_2 = X \ast k_2, \) \((k_1, k_2 \) same size): \( X \ast (\alpha k_1 + \beta k_2) = \alpha Y_1 + \beta Y_2 \)
  - If \( Y_1 = X_1 \ast k \) and \( Y_2 = X_2 \ast k, \) \((X_1, X_2 \) same size): \( (\alpha X_1 + \beta X_2) \ast k = \alpha Y_1 + \beta Y_2 \)

- **Associative**
  - \( (X \ast_{\text{full}} k_1) \ast_{\text{full}} k_2 = X \ast_{\text{full}} (k_1 \ast_{\text{full}} k_2) \)
  - \( (X \ast_{\text{val}} k_1) \ast_{\text{val}} k_2 = X \ast_{\text{val}} (k_1 \ast_{\text{full}} k_2) \)
  - \( (X \ast_{\text{same}} k_1) \ast_{\text{same}} k_2 \neq X \ast_{\text{same}} (k_1 \ast_{\text{full}} k_2) \)

\[
(X \ast_{\text{full}} k_1) \ast_{\text{full}} k_2 [n] = \sum_{n'} k_2[n'](X \ast k_1)[n-n'] = \sum_{n'} k_2[n'] \sum_{n''} k_1[n'']X[n-n'-n'']
\]

\[
= \sum_{n'+n''} \left( \sum_{n'} k_2[n']k_1[(n'+n'')-n'] \right) X[n-(n'+n'')]
\]
Let \( X *_{\text{full}} k \), \( X *_{\text{val}} k \), and \( X *_{\text{same}} k \) denote full, valid, and same convolution (with zero padding for full and same):

- **Linear / Distributive:** For scalars \( \alpha, \beta \):
  - If \( Y = X * k \), then: \( X * (\alpha k) = (\alpha X) * k = \alpha Y \)
  - If \( Y_1 = X * k_1 \) and \( Y_2 = X * k_2 \), \( (k_1, k_2 \) same size): \( X * (\alpha k_1 + \beta k_2) = \alpha Y_1 + \beta Y_2 \)
  - If \( Y_1 = X_1 * k \) and \( Y_2 = X_2 * k \), \( (X_1, X_2 \) same size): \( (\alpha X_1 + \beta X_2) * k = \alpha Y_1 + \beta Y_2 \)

- **Associative**
  - \( (X *_{\text{full}} k_1) *_{\text{full}} k_2 = X *_{\text{full}} (k_1 *_{\text{full}} k_2) \)
  - \( (X *_{\text{val}} k_1) *_{\text{val}} k_2 = X *_{\text{val}} (k_1 *_{\text{full}} k_2) \)
  - \( (X *_{\text{same}} k_1) *_{\text{same}} k_2 \neq X *_{\text{same}} (k_1 *_{\text{full}} k_2) \)

- **Commutative:** \( k_1 *_{\text{full}} k_2 = k_2 *_{\text{full}} k_1 \)
  - \( (X *_{\text{full}} k_1) *_{\text{full}} k_2 = (X *_{\text{full}} k_2) *_{\text{full}} k_1 \)
  - \( (X *_{\text{val}} k_1) *_{\text{val}} k_2 = (X *_{\text{val}} k_2) *_{\text{val}} k_1 \)
  - \( (X *_{\text{same}} k_1) *_{\text{same}} k_2 \neq (X *_{\text{same}} k_2) *_{\text{same}} k_1 \)
CONVOLUTION

\[ X[n] \]

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

\[ k[n] \]

\[ Y[n] \]

(Same with zero padding)
CONVOLUTION

\[ X[n] \]

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

\[ Y[n] \]

Kernel = Impulse Response

(Same with zero padding)

\[ k[n] \]
CONVOLUTION

\[ X[n] \quad k[n] \quad Y[n] \]

(Same with zero padding)
$X[n]$  \hspace{1.5cm}  Y[n]$

(Same with zero padding)
CONVOLUTION

$X[n]$

$Y[n]$

$k[n] = 1/25$

(Same with zero padding)
CONVOLUTION

\[ X[n] \]

\[ Y[n] \]

Gaussian Kernels

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

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

\[ \sigma = 4 \]
Unsharp Masking

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

\[ X[n] \]

\[ Y[n] \]

Unsharp Masking

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

\[ \sigma = 2 \quad \alpha = 1 \]
Unsharp Masking

\[ Y = (1 + \alpha)X - \alpha(X * G_\sigma) = X * ((1 + \alpha)\delta - \alpha G_\sigma) \]
What is an edge?

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

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

\[X \ast [1 \ 0 \ -1]\]
Derivative is a linear spatially
invariant operation: Convolution

\[X \ast \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}\]
Smoothed in y direction
"Sobel" Operator

\[X \ast \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}\]
Y Derivative
APPLICATION: EDGE DETECTION

Derivatives have been scaled so that gray (0.5) corresponds to 0. Bright to positive derivative values, dark to negative.
APPLICATION: EDGE DETECTION

Smoothing + Derivative

\[ I_x = \partial_x \ast (G_\sigma \ast X) = (\partial_x \ast G_\sigma) \ast X = G_{x:\sigma} \ast X \quad I_y = G_{y:\sigma} \ast X \]

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

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

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

Just need to convolve twice. Gives us an expression for derivative along every direction.
APPLICATION: EDGE DETECTION

Smoothing + Derivative

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

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

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

Gives us gradient magnitude and direction.

Often applied even to filters that aren't "steerable" like DoG.
APPLICATION: EDGE DETECTION

\[ I_x \]

\[ I_y \]

\[ I_{45^\circ} \]
APPLICATION: EDGE DETECTION

\[ I_x \quad I_y \quad H \]
APPLICATION: EDGE DETECTION

\[ I_x \] \hspace{1cm} \[ I_y \] \hspace{1cm} \[ H > \epsilon \]
APPLICATION: EDGE DETECTION

\( I_x \) \hspace{1cm} \( I_y \) \hspace{1cm} \( H > \epsilon \)
Extensions

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

\[ H[n] \]

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

Edges are isolated per-pixel labels
Pool them together to detect scene structure: E.g., Lines

Missed detections
Clutter
Occlusions

Edges are isolated per-pixel labels
Hough Transform

Consider ALL possible lines (on a 2D plane)
Hough Transform

- Consider ALL possible lines (on a 2D plane)
- This is a two dimensional search space, could parameterize it in different ways.

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

**Hough Transform**

Image where each pixel corresponds to some discrete value of $r$ and $\theta$

Discretize this space into a bunch of buckets.
\[ r = x \cos \theta + y \sin \theta \]

Each edge pixel casts a vote for all lines it could belong to. Compute by plugging in \( x, y \) into equation: get a value of \( r \) for each \( \theta \) (get a sinusoid)
Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x, y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)
$$r = x \cos \theta + y \sin \theta$$

**Hough Transform**

Each edge pixel casts a vote for all lines it could belong to.

Compute by plugging in $x,y$ into equation: get a value of $r$ for each $\theta$ (get a sinusoid)

Do this for all pixels and see which 'bins' get the most votes.

**Variants**
- Each edge pixel only casts one vote based on angle. (with or without sign)
- Vote weighted by magnitude of gradient.
- Exclusive vote (select dominant line, remove vote from its pixels for other lines)
- Use same idea for line segments, circles, ellipses ...