LAST TIME

**Conventions**

- 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)

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.

ADMINISTRIVIA

- Problem Set 0 Due Today!
  - Ungraded, but mandatory
- To submit, do git commit AND git push. If you don’t push, we don’t get your submission.
- You should also 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 2-3 problems.
  - Not a bad idea to use the extra time to get started early.

LAST TIME

**Point-wise Operations**

- $Y[n] = h(X[n])$
- $Y[n] = h(X_1[n], X_2[n], …)$
- $Y[n] = h_n(X[n])$ - Might vary based on location.
- $h(·)$ itself might be based on ‘global statistics’
**POINT-WISE OPERATIONS**

\[ X[n] \in \mathbb{R}^3 \]

\[
Y[n] = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} X[n] + 2.0 \quad X[n]
\]

**Linear Color Transforms**

\[ Y[n] = X[n]^{0.5} \]

**Non-linear Transforms**

**POINT-WISE OPERATIONS**

\[ X[n] \in \mathbb{R} \]

\[ Y[n] = X[n]^{2.0} \]

**Non-linear Transforms**

**Tone-maps / Change contrast**
POINT-WISE OPERATIONS

10

Y[n]

X[n]

Can be arbitrary: but usually monotonic


12

Y[n]

X[n]

POINT-WISE OPERATIONS

13

Y[n]

X[n]

Uneven distribution of intensities (many too dark or too bright)

Find a monotonic function $h(\cdot)$ such that the intensities of $h(X[n])$ are distributed uniformly in the range $[0, 255]$.

15

Histgram Equalization

People play around with this a lot manually, but also can be done automatically for some objective
POINT-WISE OPERATIONS

Consider the Cumulative Distribution (think of intensity as a continuous r.v.)

For a uniform distribution

Use the CDF as the tone map: \( h(x) = P(X[n] < x) \times 255.0 \)

POINTWISE OPERATIONS IN NUMPY

Let’s say you wanted to implement \( Y[n] = X[n]^{0.5} \).

```python
Y = np.zeros_like(X)
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        for k in range(X.shape[2]):
            Y[i,j,k] = X[i,j,k]**0.5
```

NEVER DO THIS ! Never loop through pixels. This is very inefficient.

```python
Y = X**0.5
```

This will do the same thing. Not only will you not be going through loops in the interpreter, numpy will internally use available parallelism on your CPU to do this much faster.

- In general, numpy provides a large number of point-wise functions for array. It also overloads many common arithmetic operators to these functions.
If X1 and X2 are the same shape, then Y will be the same shape and
\[ Y[i,j,k] = X1[i,j,k] + X2[i,j,k] \]
If they are not the same shape, numpy applies what is called "broadcasting". This means that you can replace the size of any dimension with 1.

X1 is (H,W,3), and X2 is (H,1,3). Then Y has shape (H,W,3):
\[ Y[i,j,k] = X1[i,j,k] + X2[i,0,k] \]
When two arrays have different numbers of dimensions, numpy assumes that any missing "leading" dimensions are 1.

X1 is (H,W,3) and X2 is (3), then X2 is "reshaped" automatically to (1,1,3).

If X1 is (H,W,3) and X2 is (H,W), then you will need to reshape X2 to size (H,W,1):
\[ Y[i,j,k] = X1[i,j,k] + X2[i,j,0] \]
If X2 is a scalar or constant, it is assumed to have shape () . It is broadcast to all dimensions, which is why
\[ X^{**0.5} \] works!

Images themselves are arrays not matrices.
But for such linear operations, we can form matrices by stacking 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.

\[ X[n] \in \mathbb{R}^{(HW) \times 3} \]
\[ X[n] \in \mathbb{R}^{(HW) \times 1} \]
\[ X \in \mathbb{R}^{(HW) \times 3} \]
\[ X \in \mathbb{R}^{(HW) \times 1} \]

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

One last trick: what if your pointwise function is a lookup table (for grayscale images). Assume that you are trying to map all intensities in an array \( X \), which is of type integer (uint8, etc.), and you have a lookup table as a vector \( H \), where you want to map an intensity \( f \) to \( H[f] \).

So what you want to do is:
\[ Y[i,j] = H[X[i,j]] \]
You can do this simply as \( Y = H[X] \). \( Y \) will be the same shape as \( X \) (and not \( H \)).
Linear operation on spatial neighborhoods

\[ Y[n] = \sum_{n'} X[n - n'] k[n'] \]
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 a 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 \( W_x, W_y \) and \( H_x \) and \( H_y \) denote the widths of \( X, k \), and \( Y \); and \( H_x, H_y \) the heights.

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

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

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

Full: Subset 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.
CONVOLUTION: PROPERTIES

Let \( \ast_k, \ast_v, \ast_s \) 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): \( (\alpha X_1 + \beta X_2) \ast k = \alpha Y_1 + \beta Y_2 \)

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

- **Associative:**
  - \( (X \ast_v k_1) \ast_v k_2 = X \ast_v (k_1 \ast_v k_2) \)
  - \( (X \ast_s k_1) \ast_s k_2 = X \ast_s (k_1 \ast_s k_2) \)
  - \( (X \ast_s k_1) \ast_s k_2 \neq X \ast_s (k_1 \ast_s k_2) \)

\[
(X \ast_s k_1) \ast_s k_2 \neq X \ast_s (k_1 \ast_s k_2)
\]
\[
= \sum_{n' \ast n''} \left( \sum_{n'} k_2 [n'] \ast \sum_{n''} k_1 [n''] \right) X[n - (n' + n'')]
\]

CONVOLUTION: PROPERTIES

Let \( \ast_v, k, \ast_s, k \), and \( X \ast_v k \) 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:**
  - \( (X \ast_v k_1) \ast_v k_2 = X \ast_v (k_1 \ast_v k_2) \)
  - \( (X \ast_s k_1) \ast_s k_2 = X \ast_s (k_1 \ast_s k_2) \)
  - \( (X \ast_s k_1) \ast_s k_2 \neq X \ast_s (k_1 \ast_s k_2) \)

\[
(X \ast_v k_1) \ast_v k_2 \neq X \ast_v (k_1 \ast_v k_2)
\]
\[
= \sum_{n' \ast n''} \left( \sum_{n'} k_2 [n'] \ast \sum_{n''} k_1 [n''] \right) X[n - (n' + n'')]
\]
CONVOLUTION

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

\[ Y'[n] \]

Kernel = Impulse Response

\[ k[n] \]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

(Same with zero padding)

\[ k[n] \]

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

(Same with zero padding)

CONVOLUTION

\[ X[n] \]

\[ Y'[n] \]

CONVOLUTION

\[ X[n] \]

\[ Y'[n] \]

\[ k[n] \]

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

(Same with zero padding)

\[ k[n] = 1/25 \]
CONVOLUTION

\( X[n] \)  \hspace{1cm} Y[n]  

Gaussian Kernels

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

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

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

\( \sigma = 2 \)

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

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

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

\( \sigma = 3 \)

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

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

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

\( \sigma = 4 \quad \alpha = 1 \)

Unsharp Masking

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

\[ Y = (1 + \alpha)X - \alpha (X \ast G_x) = X \ast \left( (1 + \alpha) \delta - \alpha G_x \right) \]