CSE 559A: Computer Vision

Fall 2019: T-R: 11:30-12:50pm @ Hillman 60

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Arushee Agrawal, Annie Lee, Jiahao Li, Xiaochen Zhou

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Sep 3, 2019
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.
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)

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

\[
\begin{bmatrix}
0.4 \\
0.4 \\
0.4 \\
\end{bmatrix} = \begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix} \begin{bmatrix}
0.1 \\
0.2 \\
0.9 \\
\end{bmatrix}
\]

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.
LAST TIME

Point-wise Operations

- $Y[n] = h(X[n])$
- $Y[n] = h(X_1[n], X_2[n], \ldots)$
- $Y[n] = h_n(X[n])$ - Might vary based on location.
- $h(\cdot)$ itself might be based on 'global statistics'
POINT-WISE OPERATIONS

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

Linear Color Transforms

\[
Y[n] = -1.0 \begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix} X[n] + 2.0 X[n]
\]
POINT-WISE OPERATIONS

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

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

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

\[ X[n] \quad \rightarrow \]

\[ Y[n] \]
POINT-WISE OPERATIONS

\[ Y[n] \]

\[ X[n] \]
Can be arbitrary: but usually monotonic


POINT-WISE OPERATIONS
People play around with this a lot manually: but also can be done automatically for some objective
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 $[0, 255]$. 

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

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

Not perfect as dealing with quantized values
POINT-WISE OPERATIONS

Image Matting (combine multiple images with alpha matte)

From Szeliski 3.1
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.

```python
Y = X1+X2
Y = np.add(X1, X2)  # Equivalent
```
If \( X_1 \) and \( X_2 \) are the same shape, then \( Y \) will be the same shape and
\[
Y[i,j,k] = X_1[i,j,k] + X_2[i,j,k]
\]

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.

\( X_1 \) is \((H,W,3)\), and \( X_2 \) is \((H,1,3)\). Then \( Y \) has shape \((H,W,3)\):
\[
Y[i,j,k] = X_1[i,j,k] + X_2[i,0,k]
\]

When two arrays have different numbers of dimensions, numpy assumes that any missing **leading** dimensions are 1.

\( X_1 \) is \((H,W,3)\) and \( X_2 \) is \((3)\), then \( X_2 \) is "reshaped" automatically to \((1,1,3)\).

If \( X_1 \) is \((H,W,3)\) and \( X_2 \) is \((H,W)\), then you will need to reshape \( X_2 \) to size \((H,W,1)\):
\[
Y = X_1+X_2[:,:,np.newaxis]
\]

If \( X_2 \) is a scalar or constant, it is assumed to have shape \((1)\). It is broadcast to all dimensions, which is why \( X**0.5 \) works!
Your goal, try to express pointwise operations in a way that expresses it as a combination of parallel numpy operations.

For example, for alpha matting, let’s say you have two images $X_1$ and $X_2$ both of size $(H,W,3)$, and a mask of size $(H,W)$ that goes from 0-1 (1 indicating keep $X_1$, and 0 indicating keep $X_2$):

$$Y = X_1 \times \text{mask[:, :, np.newaxis]} + X_2 \times (1. - \text{mask[:, :, np.newaxis]})$$

But what about the linear matrix multiplies we talked about?

Not that the * operator, which corresponds to `numpy.mul`, does element-wise multiply.

`numpy.matmul` does matrix multiplies, but what we want are matrix multiplies applied to each pixel's color vector.
POINTWISE OPERATIONS IN NUMPY

- 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.

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

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

Linear operation on spatial neighborhoods

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

Kernel
CONVOLUTION
CONVOLUTION
CONVOLUTION
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": \( k[n] \in \mathbb{R}^{n \times m} \) 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
What is the range / size of $Y$?

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

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

$\begin{align*}
X & \quad W_x \\
H_x & \\
Y_{[W_k-1, H_k-1]} & \quad W_y \\
H_y & \\
Y & \\
W_y & = W_x - W_k + 1 \\
H_y & = H_x - H_k + 1
\end{align*}$

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

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

\[ (0,0) \]

\[ \begin{array}{c}
W_k \\
H_k
\end{array} \]

\[ X \]

\[ Y \]

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

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

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

\[ X[n] \]

\[ Y[n] \]

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

\[ k[n] \]

(Same with zero padding)
CONVOLUTION

$X[n]$

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$Y[n]$

Kernel = Impulse Response

(Same with zero padding)

$k[n]$
$X[n]$  

![Image of dog](image1.png)

$Y[n]$  

![Image of dog](image2.png)

$k[n]$

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\] (Same with zero padding)
CONVOLUTION

\[ X[n] \]

\[ Y[n] \]

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

\[ k[n] \]

(Same with zero padding)
CONVOLUTION

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

\[
\begin{array}{c}
\text{(Same with zero padding)}
\end{array}
\]

\[ k[n] = \frac{1}{25} \]
CONVOLUTION

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

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

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

\[ \sigma = 2 \]
Gaussian Kernels

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

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

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

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

Gaussian Kernels

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

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

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

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

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