WHAT IS AN IMAGE?
>>> from numpy import random as r
>>> I = r.rand(256,256);

Think-Pair-Share:
- What is this? What does it look like?
- Which values does it take?
- How many values can it take?
- Is it an image?
```python
>>> from matplotlib import pyplot as p
>>> I = r.rand(256,256);
>>> p.imshow(I);
>>> p.show();
```
Dimensionality of an Image

• @ 8bit = 256 values ^ 65,536
  – Computer says ‘Inf’ combinations.

• Some depiction of all possible scenes would fit into this memory.
Dimensionality of an Image

• @ 8bit = 256 values ^ 65,536
  – Computer says ‘Inf’ combinations.

• Some depiction of all possible scenes would fit into this memory.

• Computer vision as making sense of an extremely high-dimensional space.
  – Subspace of ‘natural’ images.
  – Deriving low-dimensional, explainable models.
What is each part of an image?
What is each part of an image?

- Pixel -> picture element

\[ I(x, y) \]
Image as a 2D sampling of signal

- Signal: function depending on some variable with physical meaning.

- Image: sampling of that function.
  - 2 variables: $xy$ coordinates
  - 3 variables: $xy +$ time (video)
  - ‘Brightness’ is the value of the function for visible light

- Can be other physical values too: temperature, pressure, depth ...
Example 2D Images
Sampling in 1D

- Sampling in 1D takes a function, and returns a vector whose elements are values of that function at the sample points.
Sampling in 2D

• Sampling in 2D takes a function and returns a matrix.
Grayscale Digital Image

Brightness or intensity

Danny Alexander
What is each part of a photograph?

- Pixel -> picture element

Where $I(x, y)$ is a pixel at coordinates $(x, y)$ with the value '127'.
Integrating light over a range of angles.
Resolution – geometric vs. spatial resolution

Both images are ~500x500 pixels
Quantization
Quantization Effects – Radiometric Resolution

- 8 bit – 256 levels
- 4 bit – 16 levels
- 2 bit – 4 levels
- 1 bit – 2 levels
Images in Python Numpy

N x M grayscale image “im”
- \(im[0,0] = \) top-left pixel value
- \(im[y, x] = \) y pixels down, x pixels to right
- \(im[N-1, M-1] = \) bottom-right pixel

\[
\begin{array}{cccccccccccc}
0.92 & 0.93 & 0.94 & 0.97 & 0.62 & 0.37 & 0.85 & 0.97 & 0.93 & 0.92 & 0.99 \\
0.95 & 0.89 & 0.82 & 0.89 & 0.56 & 0.31 & 0.75 & 0.92 & 0.81 & 0.95 & 0.91 \\
0.89 & 0.72 & 0.51 & 0.55 & 0.51 & 0.42 & 0.57 & 0.41 & 0.49 & 0.91 & 0.92 \\
0.96 & 0.95 & 0.88 & 0.94 & 0.56 & 0.46 & 0.91 & 0.87 & 0.90 & 0.97 & 0.95 \\
0.71 & 0.81 & 0.81 & 0.87 & 0.57 & 0.37 & 0.80 & 0.88 & 0.89 & 0.79 & 0.85 \\
0.49 & 0.62 & 0.60 & 0.58 & 0.50 & 0.60 & 0.58 & 0.50 & 0.61 & 0.45 & 0.33 \\
0.86 & 0.84 & 0.74 & 0.58 & 0.51 & 0.39 & 0.73 & 0.92 & 0.91 & 0.49 & 0.74 \\
0.96 & 0.67 & 0.54 & 0.85 & 0.48 & 0.37 & 0.88 & 0.90 & 0.94 & 0.82 & 0.93 \\
0.69 & 0.49 & 0.56 & 0.66 & 0.43 & 0.42 & 0.77 & 0.73 & 0.71 & 0.90 & 0.99 \\
0.79 & 0.73 & 0.90 & 0.67 & 0.33 & 0.61 & 0.69 & 0.79 & 0.73 & 0.93 & 0.97 \\
0.91 & 0.94 & 0.89 & 0.49 & 0.41 & 0.78 & 0.78 & 0.77 & 0.89 & 0.99 & 0.93 \\
\end{array}
\]
Grayscale intensity

<table>
<thead>
<tr>
<th>0.92</th>
<th>0.93</th>
<th>0.94</th>
<th>0.97</th>
<th>0.62</th>
<th>0.37</th>
<th>0.85</th>
<th>0.97</th>
<th>0.93</th>
<th>0.92</th>
<th>0.99</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.89</td>
<td>0.82</td>
<td>0.89</td>
<td>0.56</td>
<td>0.31</td>
<td>0.75</td>
<td>0.92</td>
<td>0.81</td>
<td>0.95</td>
<td>0.91</td>
</tr>
<tr>
<td>0.89</td>
<td>0.72</td>
<td>0.51</td>
<td>0.55</td>
<td>0.51</td>
<td>0.42</td>
<td>0.57</td>
<td>0.41</td>
<td>0.49</td>
<td>0.91</td>
<td>0.92</td>
</tr>
<tr>
<td>0.96</td>
<td>0.95</td>
<td>0.88</td>
<td>0.94</td>
<td>0.56</td>
<td>0.46</td>
<td>0.91</td>
<td>0.87</td>
<td>0.90</td>
<td>0.97</td>
<td>0.95</td>
</tr>
<tr>
<td>0.71</td>
<td>0.81</td>
<td>0.81</td>
<td>0.87</td>
<td>0.57</td>
<td>0.37</td>
<td>0.80</td>
<td>0.88</td>
<td>0.89</td>
<td>0.79</td>
<td>0.85</td>
</tr>
<tr>
<td>0.49</td>
<td>0.62</td>
<td>0.60</td>
<td>0.58</td>
<td>0.50</td>
<td>0.60</td>
<td>0.58</td>
<td>0.50</td>
<td>0.61</td>
<td>0.45</td>
<td>0.33</td>
</tr>
<tr>
<td>0.86</td>
<td>0.84</td>
<td>0.74</td>
<td>0.58</td>
<td>0.51</td>
<td>0.39</td>
<td>0.73</td>
<td>0.92</td>
<td>0.91</td>
<td>0.49</td>
<td>0.74</td>
</tr>
<tr>
<td>0.96</td>
<td>0.67</td>
<td>0.54</td>
<td>0.85</td>
<td>0.48</td>
<td>0.37</td>
<td>0.88</td>
<td>0.90</td>
<td>0.94</td>
<td>0.82</td>
<td>0.93</td>
</tr>
<tr>
<td>0.69</td>
<td>0.49</td>
<td>0.56</td>
<td>0.66</td>
<td>0.43</td>
<td>0.42</td>
<td>0.77</td>
<td>0.73</td>
<td>0.71</td>
<td>0.90</td>
<td>0.99</td>
</tr>
<tr>
<td>0.79</td>
<td>0.73</td>
<td>0.90</td>
<td>0.67</td>
<td>0.33</td>
<td>0.61</td>
<td>0.69</td>
<td>0.79</td>
<td>0.73</td>
<td>0.93</td>
<td>0.97</td>
</tr>
<tr>
<td>0.91</td>
<td>0.94</td>
<td>0.89</td>
<td>0.49</td>
<td>0.41</td>
<td>0.78</td>
<td>0.78</td>
<td>0.77</td>
<td>0.89</td>
<td>0.99</td>
<td>0.93</td>
</tr>
</tbody>
</table>
Color

James Hays
Images in Python Numpy

N x M RGB image “im”

- \( \text{im}[0,0,0] \) = top-left pixel value in R-channel
- \( \text{im}[x, y, b] \) = x pixels to right, y pixels down in the b\(^{th}\) channel
- \( \text{im}[N-1, M-1, 3] \) = bottom-right pixel in B-channel
Images in Python Numpy

Take care between types!

- **uint8**  (values 0 to 255)  – io.imread(“file.jpg”)
- **float32**  (values 0 to 255)  – io.imread(“file.jpg”).astype(np.float32)
- **float32**  (values 0 to 1)  – img_as_float32(io.imread(“file.jpg”))
IMAGE FILTERING
Image filtering

• Image filtering:
  – Compute function of local neighborhood at each position

\[ h[m, n] = \sum_{k,l} f[k, l] I[m+k, n+l] \]
Image filtering

- Image filtering:
  - Compute function of local neighborhood at each position

\[
h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l]
\]

\[
2d \text{ coords}=k, l \quad 2d \text{ coords}=m, n
\]
Example: box filter

\[ f[\cdot, \cdot] \]

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\frac{1}{9}
\]
Image filtering

\[ I[\cdot,\cdot] \]

\[ f[\cdot,\cdot] = \frac{1}{9} \]

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \]

\[ m = 1, n = 1 \]

\[ k, l = [-1,0,1] \]

Credit: S. Seitz
Image filtering

\[ I[\cdot,\cdot] \]

\[ h[\cdot,\cdot] \]

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \]

\[ m = 2, n = 1 \]

\[ k, l = [-1,0,1] \]
Image filtering

\[
I[\cdot, \cdot]
\]

\[
h[\cdot, \cdot]
\]

\[
h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]
\]

\[
f[\cdot, \cdot] \frac{1}{9}
\]

\[
m = 3, n = 1
\]

\[
k, l = [-1, 0, 1]
\]

Credit: S. Seitz
Image filtering

\[ I[\cdot,\cdot] \]

\[ h[\cdot,\cdot] \]

\[ f[\cdot,\cdot] \]

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \]

\[ m = 4, n = 1 \]

\[ k, l = [-1,0,1] \]

Credit: S. Seitz
Image filtering

\[ f[\cdot, \cdot] \]

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

\[ h[m, n] = \sum_{k,l} f[k, l] I[m+k, n+l] \]

\[ m = 5, n = 1 \]

\[ k, l = [-1,0,1] \]

Credit: S. Seitz
Image filtering

\[ f[\cdot, \cdot] = \frac{1}{9} \]

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

\[ h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l] \]

\[ m = 4, n = 6 \]

\[ k, l = [-1, 0, 1] \]

Credit: S. Seitz
Image filtering

\[ I[\cdot,\cdot] \]

\[ h[\cdot,\cdot] \]

\[
h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \quad m = 6, n = 4
\]

\[ k, l = [-1,0,1] \]
Image filtering

\[ I[\cdot, \cdot] \]

\[ h[\cdot, \cdot] = \sum_{k,l} f[k, l] I[m + k, n + l] \]
What does it do?

• Replaces each pixel with an average of its neighborhood

• Achieve smoothing effect (remove sharp features)

Slide credit: David Lowe (UBC)
What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)
- Why does it sum to one?

\[
f[\cdot,\cdot]
\]

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

Slide credit: David Lowe (UBC)
Smoothing with box filter

\[
f[\cdot, \cdot] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}
\]
Image filtering

• Image filtering:
  – Compute function of local neighborhood at each position

\[
h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]
\]

• Really important!
  – Enhance images
    • Denoise, resize, increase contrast, etc.
  – Extract information from images
    • Texture, edges, distinctive points, etc.
  – Detect patterns
    • Template matching
Think-Pair-Share time

1. 
   0 0 0
   0 1 0
   0 0 0

2. 
   0 0 0
   0 0 1
   0 0 0

3. 
   1 0 -1
   2 0 -2
   1 0 -1

4. 
   0 0 0
   0 2 0
   0 0 0

\[-\frac{1}{9}\]
1. Practice with linear filters

Original

Source: D. Lowe
1. Practice with linear filters

Original

Filtered (no change)

Source: D. Lowe
2. Practice with linear filters

Original

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
2. Practice with linear filters

Original

Shifted left
By 1 pixel

Source: D. Lowe
3. Practice with linear filters

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Sobel

Vertical Edge (absolute value)

David Lowe
3. Practice with linear filters

Sobel

1 2 1
0 0 0
-1 -2 -1

Horizontal Edge (absolute value)
4. Practice with linear filters

Original

(Note that filter sums to 1)
4. Practice with linear filters

**Sharpening filter**
- Accentuates differences with local average

Source: D. Lowe
4. Practice with linear filters

Source: D. Lowe
Correlation and Convolution

• 2d correlation

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \]

e.g., \( h = \text{scipy.signal.correlate2d}(f,I) \)
Correlation and Convolution

• 2d correlation

\[
h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l]
\]

E.g., \( h = \text{scipy.signal.correlate2d}(f,I) \)

• 2d convolution

\[
h[m,n] = \sum_{k,l} f[k,l] I[m-k,n-l]
\]

E.g., \( h = \text{scipy.signal.convolve2d}(f,I) \)

Convolution is the same as correlation with a 180° rotated filter kernel. Correlation and convolution are identical when the filter kernel is symmetric.
Key properties of linear filters

**Linearity:**
\[
\text{imfilter}(I, f_1 + f_2) = \text{imfilter}(I, f_1) + \text{imfilter}(I, f_2)
\]

**Shift invariance:**
Same behavior given intensities regardless of pixel location \(m,n\)
\[
\text{imfilter}(I, \text{shift}(f)) = \text{shift}(\text{imfilter}(I, f))
\]

Any linear, shift-invariant operator can be represented as a convolution.
Convolution properties

Commutative: $a \ast b = b \ast a$
- Conceptually no difference between filter and signal
- But particular filtering implementations might break this equality, e.g., image edges

Associative: $a \ast (b \ast c) = (a \ast b) \ast c$
- Often apply several filters one after another: $(((a \ast b_1) \ast b_2) \ast b_3)$
- This is equivalent to applying one filter: $a \ast (b_1 \ast b_2 \ast b_3)$

Source: S. Lazebnik
Convolution properties

Commutative: $a * b = b * a$
- Conceptually no difference between filter and signal
- But particular filtering implementations might break this equality, e.g., image edges

Associative: $a * (b * c) = (a * b) * c$
- Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
- This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Correlation is _not_ associative (rotation effect)
- Why important?

Source: S. Lazebnik
Recap of Monday

• Linear filtering (convolution)

\[ h[m, n] = \sum_{k,l} f[k, l] I[m - k, n - l] \]

– Not a matrix multiplication
– Sum over Hadamard product
– Can smooth, sharpen, translate
  (among many other uses)
Convolution properties

• **Commutative:** $a * b = b * a$
  – Conceptually no difference between filter and signal
  – But particular filtering implementations might break this equality, e.g., image edges

• **Associative:** $a * (b * c) = (a * b) * c$
  – Often apply several filters one after another: ((($a * b_1) * b_2) * b_3)
  – This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
  – Correlation is _not_ associative (rotation effect)
  – Why important?

• **Distributes over addition:** $a * (b + c) = (a * b) + (a * c)$

• **Scalars factor out:** $k a * b = a * k b = k (a * b)$

• **Identity:** unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$

Source: S. Lazebnik
Important filter: Gaussian

Weight contributions of neighboring pixels by nearness

\[ G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \]
Smoothing with Gaussian filter
Smoothing with box filter
Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
  - Images become more smooth
- Gaussian convolved with Gaussian...
  ...is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving twice with Gaussian kernel of width $\sigma$ is same as convolving once with kernel of width $\sigma\sqrt{2}$

- *Separable* kernel
  - Factors into product of two 1D Gaussians
Separability of the Gaussian filter

\[ G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left( -\frac{x^2 + y^2}{2\sigma^2} \right) \]

\[ = \left( \frac{1}{\sqrt{2\pi\sigma}} \exp\left( -\frac{x^2}{2\sigma^2} \right) \right) \left( \frac{1}{\sqrt{2\pi\sigma}} \exp\left( -\frac{y^2}{2\sigma^2} \right) \right) \]

The 2D Gaussian can be expressed as the product of two functions, one a function of \( x \) and the other a function of \( y \).

In this case, the two functions are the (identical) 1D Gaussian.
Separability example

2D convolution (center location only)

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix} * \begin{bmatrix}
2 & 3 & 3 \\
3 & 5 & 5 \\
4 & 4 & 6 \\
\end{bmatrix}
\]

The filter factors into a product of 1D filters:

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}
\]

Perform convolution along rows:

\[
\begin{bmatrix}
1 & 2 & 1 \\
3 & 5 & 5 \\
4 & 4 & 6 \\
\end{bmatrix} *
\begin{bmatrix}
1 & 2 & 1 \\
3 & 5 & 5 \\
4 & 4 & 6 \\
\end{bmatrix}
\]

Followed by convolution along the remaining column:

\[
\begin{bmatrix}
11 \\
18 \\
18 \\
\end{bmatrix}
\]

Source: K. Grauman
Separability

Why is separability useful in practice?

MxN image, PxQ filter

- 2D convolution: $\sim MNPQ$ multiply-adds
- Separable 2D: $\sim MN(P+Q)$ multiply-adds

Speed up = $PQ/(P+Q)$
9x9 filter = $\sim 4.5x$ faster
How big should the filter be?

• Values at edges should be near zero
• Gaussians have infinite extent...
• Rule of thumb for Gaussian: set filter half-width to about 3 $\sigma$
Practical matters

What about near the edge?

– The filter window falls off the edge of the image
– Need to extrapolate
– methods:
  • clip filter (black)
  • wrap around
  • copy edge
  • reflect across edge

Source: S. Marschner
Convolution in Convolutional Neural Networks

• Convolution is the basic operation in CNNs
• Learning convolution kernels allows us to learn which `features’ provide useful information in images.
Sobel filter visualization

• What happens to negative numbers?

• For visualization:
  – Shift image + 0.5
  – If gradients are small, scale edge response
```python
>> I = img_to_float32( io.imread('luke.jpg') );
>> h = convolve2d( I, sobelKernel );

plt.imshow(h + 0.5); plt.imshow(h);
```

Sobel kernel:
```
1  2  1
0  0  0
-1 -2 -1
```
$h(:, :, 1) < 0$  \hspace{2cm}  $h(:, :, 1) > 0$
Think-Pair-Share

a) \( \_ = D \ast B \)
b) \( A = \_ \ast \_ \)
c) \( F = D \ast \_ \)
d) \( \_ = D \ast D \)

* = Convolution operator

A

B

C

D

E

F

G

H

I

Hoiem
\[ I = D \ast D \]

“...something to do with lack of content (black) at edges...”
\[ I = D \times D \]

```python
>> D = img_to_float32( io.imread('convexample.png') )
>> I = convolve2d( D, D )
>> np.max(I)
1.1021e+04

# Normalize for visualization
>> I_norm = (I - np.min(I)) / (np.max(I) - np.min(I))
>> plt.imshow(I_norm)
```
\[ I = D \times D \]

For x: \(275 + \frac{(275-1)}{2} + \frac{(275-1)}{2} = 549\)
\[ I = D \times D \]

>> I = convolve2d(D, D, mode='full')

(Default; pad with zeros)

>> I = convolve2d(D, D, mode='same')

(Return same size as D)

>> I = convolve2d(D, D, mode='valid')

(No padding)

Value = 10528.3
\[ A = B \ast C \]  

“because it kind of looks like it.”

When the filter ‘looks like’ the image = ‘template matching’

Filtering viewed as comparing an image of what you want to find against all image regions.

For symmetric filters: use either convolution or correlation.  
For nonsymmetric filters: correlation is template matching.
Filtering: Correlation and Convolution

• 2d correlation

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \]

e.g., \( h = \text{scipy.signal.correlate2d}(f, I) \)

• 2d convolution

\[ h[m,n] = \sum_{k,l} f[k,l] I[m-k,n-l] \]

e.g., \( h = \text{scipy.signal.convolve2d}(f, I) \)

Convolution is the same as correlation with a 180° rotated filter kernel. Correlation and convolution are identical when the filter kernel is symmetric.
OK, so let’s test this idea. Let’s see if we can use correlation to ‘find’ the parts of the image that look like the filter.

```
>> f = D[57:117, 107:167]

Expect response ‘peak’ in middle of I
```

```
>> I = correlate2d( D, f, 'same' )
```

Hmm…
That didn’t work – why not?
Correlation

\[ h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l] \]

e.g., \( h = \text{scipy.signal.correlate2d}(f,I) \)

As brightness in \( I \) increases, the response in \( h \) will increase, as long as \( f \) is positive.

**Overall brighter regions will give higher correlation response -> not useful!**
OK, so let’s subtract the mean

```
>> f = D[57:117, 107:167]
>> f2 = f - np.mean(f)
>> D2 = D - np.mean(D)
```

Now zero centered.  
*Score is higher only when dark parts match and when light parts match.*

```
>> I2 = correlate2d( D2, f2, 'same' )
```
Or even

$$\text{>> I3 = correlate2d( D2, D2, 'full' )}$$

D2 (275 x 175 pixels)
What happens with convolution?

```python
>> f = D[57:117, 107:167]
>> f2 = f - np.mean(f)
>> D2 = D - np.mean(D)
```
NON-LINEAR FILTERS
Median filters

• Operates over a window by selecting the median intensity in the window.
• ‘Rank’ filter as based on ordering of gray levels
  – E.G., min, max, range filters
Image filtering - mean

\[ I[\cdot,\cdot] \]

\[ h[\cdot,\cdot] = \sum_{k,l} f[k,l] I[m+k,n+l] \]
Image filtering - mean

$$I[\cdot,\cdot]$$

$$h[\cdot,\cdot]$$

$$h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l]$$

Credit: S. Seitz
Median filter?

$I[.,.]$

$h[.,.]$

Credit: S. Seitz
Median filters

• Operates over a window by selecting the median intensity in the window.
• What advantage does a median filter have over a mean filter?
Noise – Salt and Pepper Jack
Mean Jack – 3 x 3 filter
Very Mean Jack – 11 x 11 filter
Noisy Jack – Salt and Pepper
Median Jack – 3 x 3
Median filters

• Operates over a window by selecting the median intensity in the window.
• What advantage does a median filter have over a mean filter?
• Is a median filter a kind of convolution?
Median filters

• Operates over a window by selecting the median intensity in the window.
• What advantage does a median filter have over a mean filter?
• Is a median filter a kind of convolution?

Interpretation: Median filtering is sorting.
Tilt-shift photography
Tilt shift camera

Shift

Tilt

Plane of focus

Normal lens

Tilted Lens

Sensor

Plane of focus
Macro photography

Key:
- Lens Plane (LP)
- Image Plane (IP)
- Focal volume

Small scale in-focus volume

© Bhavuk Joshi 2016. All rights reserved.
Can we fake tilt shift?

• We need to blur the image
  – OK, now we know how to do that.
Can we fake tilt shift?

• We need to blur the image
  – OK, now we know how to do that.

• We need to blur progressively more away from our ‘fake’ focal point
But can I make it look more like a toy?

- Boost saturation – toys are very colorful
- We’ll learn how to do this when we discuss color
- For now: transform to Hue, Saturation, Value instead of RGB
Next class: Thinking in Frequency