Image Filtering and Segmentation

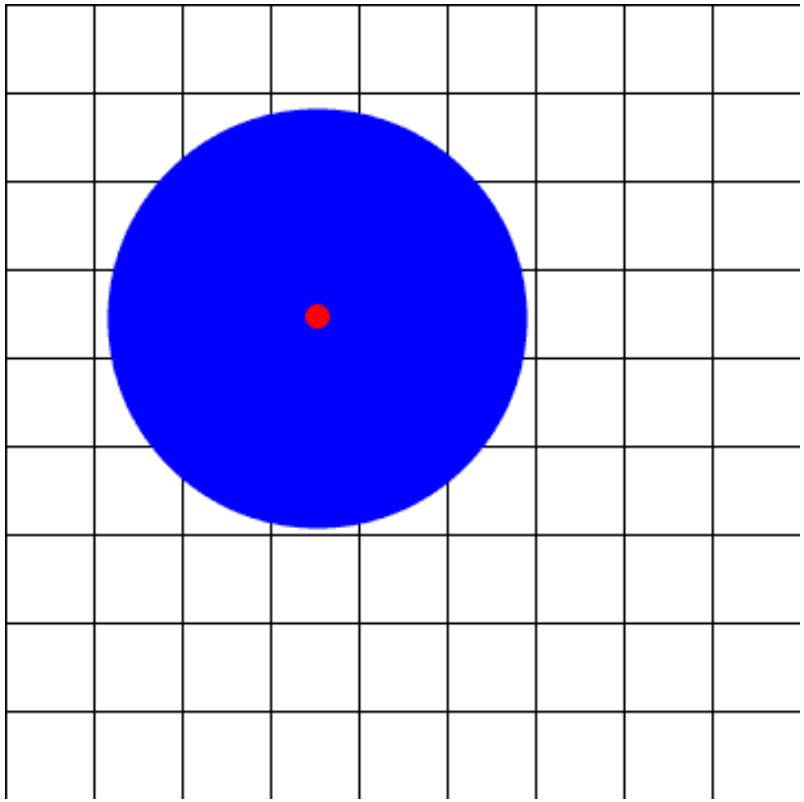
Image Filtering

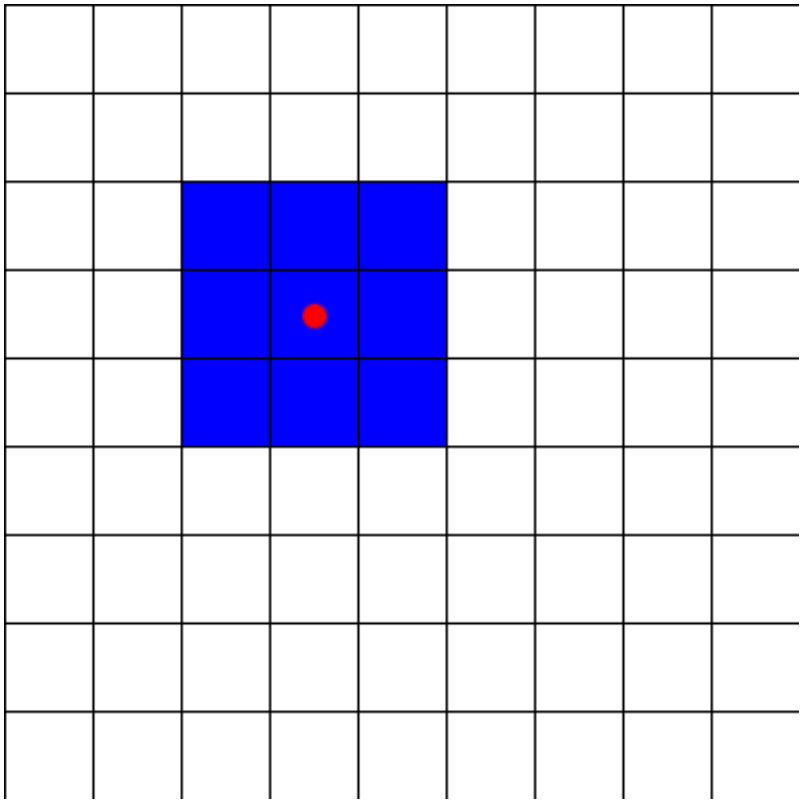
- Uses
  - Remove noise
  - Sharpen contrast
  - Highlight contours
  - Detect edges
  - Morphing
  - Segmentation

Window-Based Filtering

- The output at a location is based on the input in a neighbourhood around the location
- This can be thought of as a window/mask around the image
  - For each pixel in the image, the window is overlaid at that location

Neighbourhood





Since pixels are rectangular, neighbourhood is usually square

Median Filtering

- The output is the median of the neighbourhood area.
- $\text{out}(x, y) = \text{median}(\ \text{neighbourhood}(\text{in}(x, y))\ )$

- Effective at noise removal
  - Preserves edge

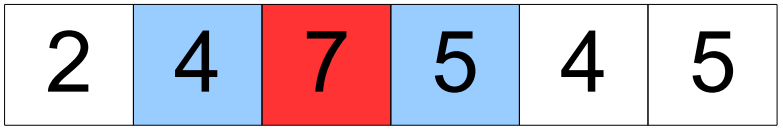
Median

- One of many “averages” in statistics
  - Others include *Mean*, *Mode*, etc.
- Given a list of number
- Median is the “central” number
  - With *N* numbers in an array
  - First sort them
  - Median is the number at index *N / 2*

Example

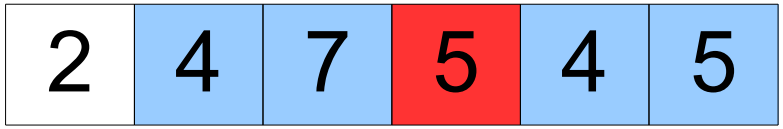
- 1D

1 x 3

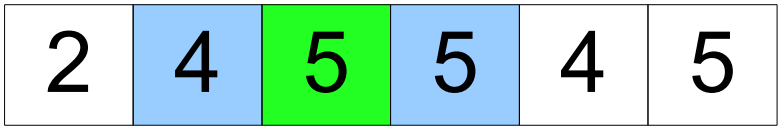


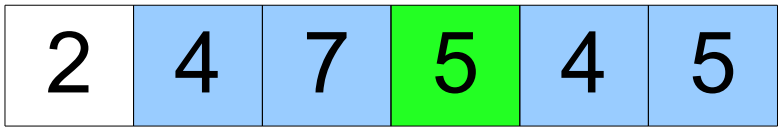
Median(4, 7, 5) = 5

1 x 5



Median(4, 7, 5, 4, 5) = 5





Example

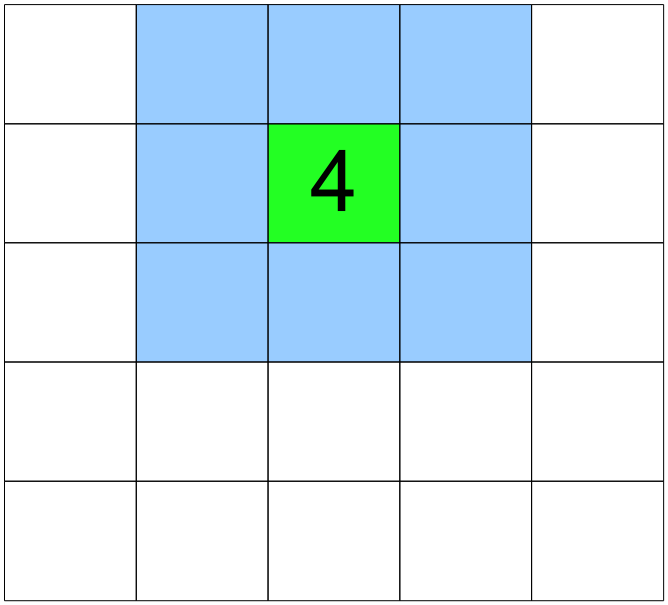
- 2D

1	6	5	6	2
7	4	0	4	2
3	2	4	3	7
6	6	1	5	7
0	6	4	2	0

Median(
6, 5, 6, 4, 0, 4, 2, 4, 3
) = ?

Sort

Median(
0, 2, 3, 4, **4**, 4, 5, 6, 6
) = 4



Implementation in ImageJ

Part *pseudo-code*:

```
window_area; // somehow get the window area (probably user defined or constant)

foreach pixel (x,y) {

    // make an array
    neighbourhood = newArray(window_area);

    // somehow fill the array with values in neighbourhood at position (x,y)

    Array.sort(neighbourhood);

    output = neighbourhood[ floor(window_area / 2) ];
    // assuming window_area is odd, then floor() or integer division will give the correct result.
}
```

Border Issues

- The neighbourhood is not well defined at the border of the image.

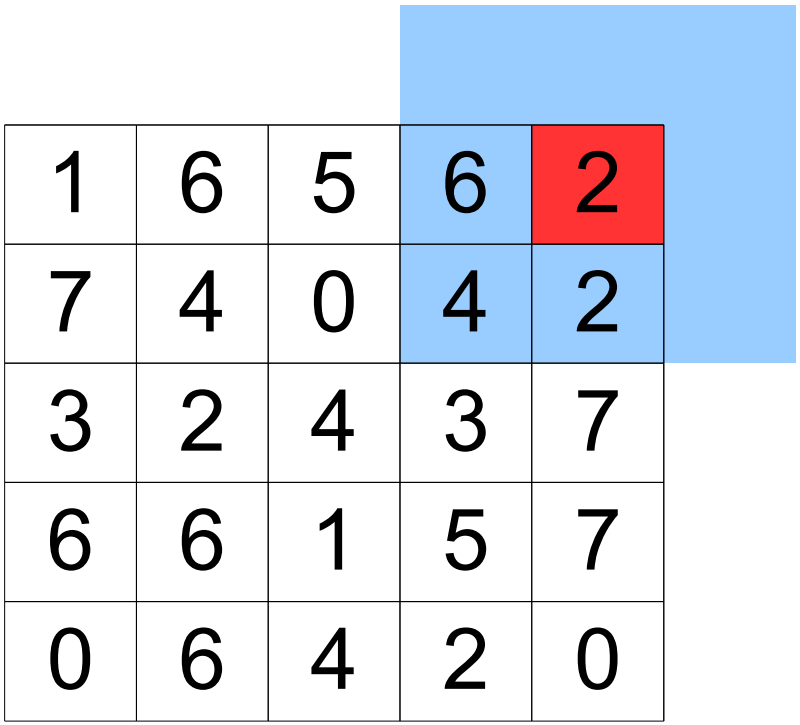
			?	?	?
1	6	5	6	2	?
7	4	0	4	2	?
3	2	4	3	7	
6	6	1	5	7	
0	6	4	2	0	

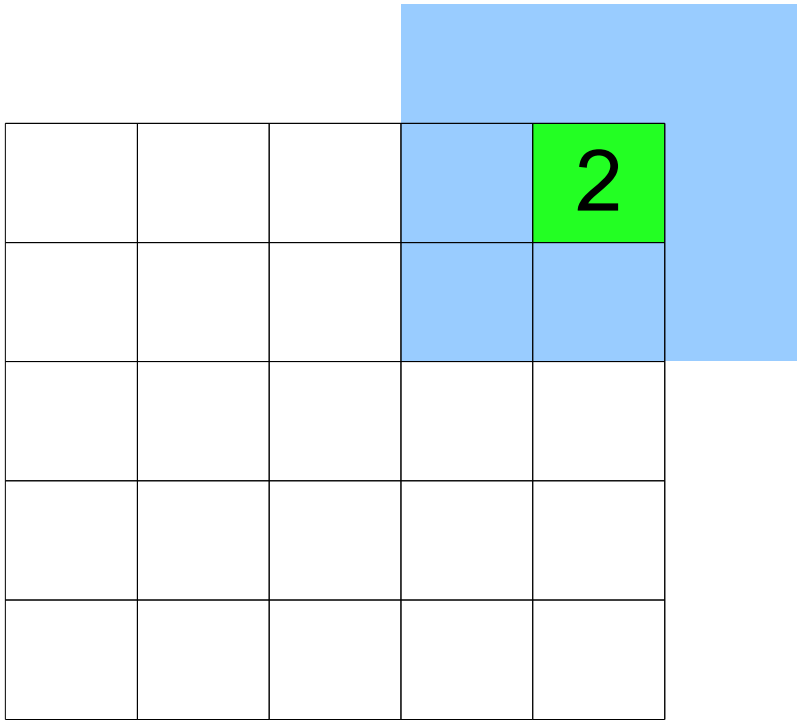
Border Issues

- Different methods exist to solve border issue
  - Ignore
    - Zero padding
    - Keep current
  - Truncate
    - Truncate (or transform) neighbourhood
    - Truncate image (resize)
  - Repeat border
  - Mirroring (Reflected indexing)
  - Circular indexing
- In this course, we'll use the *keep current, repeat border* or *zero padding* method for simplicity.

Border Issues

- Keep Current
  - At the border where the neighbourhood is ill-defined
  - Copy the input value to the output unchanged





Border Issues

- Keep Current

1	6	5	6	2
7	4	0	4	2
3	2	4	3	7
6	6	1	5	7
0	6	4	2	0

1	6	5	6	2
7				2
3				7
6				7
0	6	4	2	0

Border Issues

- Repeat Border
  - When at border
  - Fill out-of-bound position with the value at the border

1	6	5	6	2
7	4	0	4	2
3	2	4	3	7
6	6	1	5	7
0	6	4	2	0

			6	2	2
1	6	5	6	2	2
7	4	0	4	2	2
3	2	4	3	7	
6	6	1	5	7	
0	6	4	2	0	

Border Issues

- Repeat Border

1	6	5	6	2
7	4	0	4	2
3	2	4	3	7
6	6	1	5	7
0	6	4	2	0

1	6	5	6	2
7	4	0	4	2
3	2	4	3	7
6	6	1	5	7
0	6	4	2	0
		4	2	0

Average Filtering

- Uses the *mean* instead of median as the operation
- $\text{out}(x, y) = \text{mean}(\ \text{neighbourhood}(\text{in}(x, y))\ )$
- Blurs the image

Example

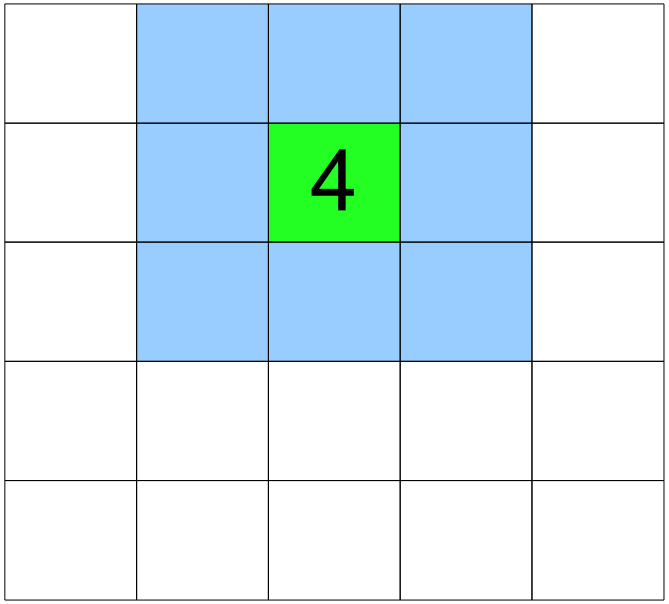
- 2D

1	6	5	6	2
7	4	0	4	2
3	2	4	3	7
6	6	1	5	7
0	6	4	2	0

Mean(
6, 5, 6, 4, 0, 4, 2, 4, 3
) = ?

(6+5+6+4+0+4+2+4+3)/9
= 3.77778

round(3.77778) = 4



- Compare different border methods to OpenGL texture settings
  - repeat border = GL_CLAMP
  - circular indexing = GL_REPEAT

Image Segmentation

- Splits images into different regions
- For binary segmentation, typically label regions using binary mask

Simple Thresholding

- Uses a single threshold (typically manually defined)

- $out(x, y) = (\text{in}(x, y) < \tau)\ ?\ 0 : I_{max}$

Average Thresholding

- Computes the average (mean) value of the image as the threshold
- Either
  - Sum over all pixels and divide by image size
  - Or
  - Multiply intensity by the normalized histogram

$$\sum_{i=0}^{255} i * \tilde{m}$$

Adaptive Thresholding

- Attempts to compute the best separation value of two peaks in the image histogram
- Iterative algorithm, we repeat the process until the value converges

Adaptive Thresholding

1. Start with some threshold value T
2. Compute the average intensity *u* of each class, so we get u_{ob} and u_{bg}
3. Get a new threshold as the mean of the two *u* of each class $T_{j+1} = \frac{\mu_{j,ob} + \mu_{j,bg}}{2}$
4. Repeat until the new threshold is not changing

$$T_{j+1} = T_j$$

Computing Average

- If we have the histogram of an image, we can compute the average value for some intensity range quickly
- The average value for some intensity range [a,b] is $$\frac{\sum_{i=a}^{b} i * m}{\sum_{i=a}^{b} m}$$

Region Growing

- Gets a continuous region around a seed point based on colour
  - Similar to the “magic wand” tool in Photoshop
- Commonly the predicate is a simple intensity range compared to the seed point or the current pixel
  - If the seed point have colour c and our tolerance is r, we get all connected pixels with colour between c-r and c+r

Region Growing

- Can be easily implemented using a variation of breadth-first-search (BFS)
- Start from a seed point, add all un-visited neighbours (4 or 8) that satisfies the predicate to the queue and mark current location as visited
- Dequeue and repeat until queue empty