Announcements

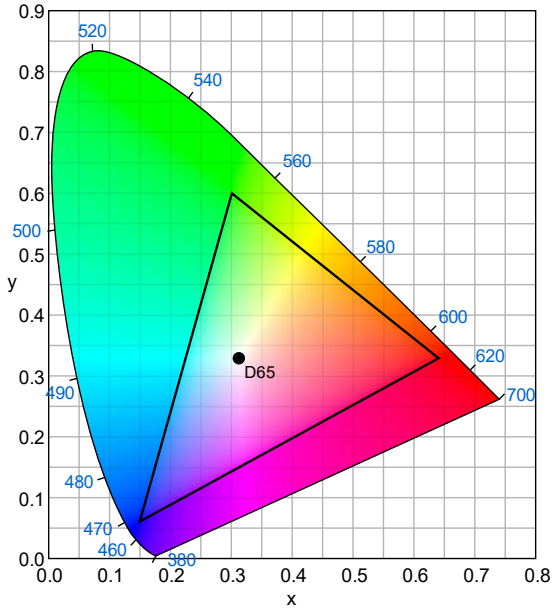
- A3 due Oct 18
- Last lecture Oct 18: function handles

Today's topic: image processing

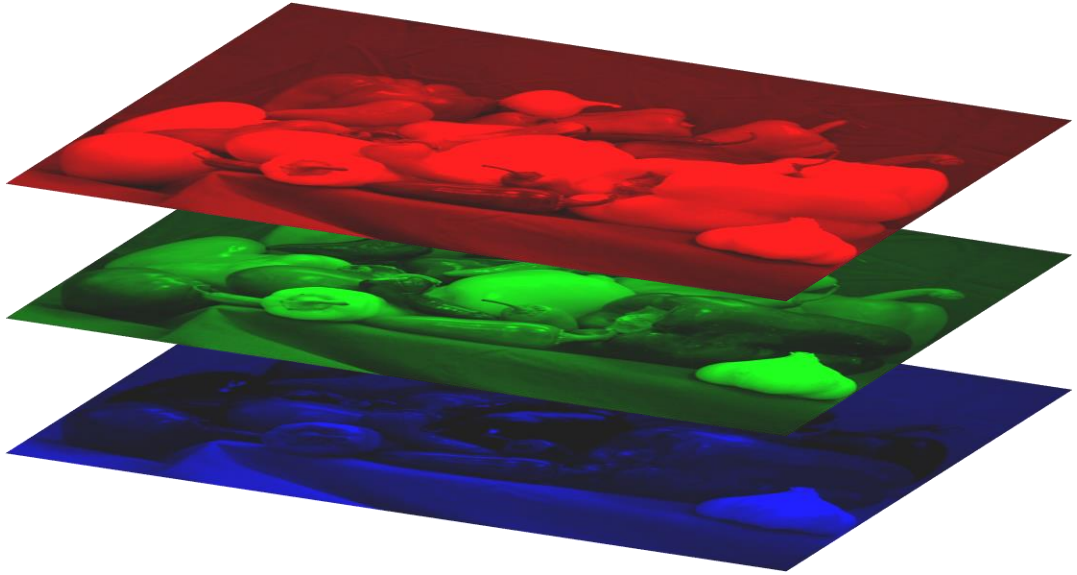
- 3D arrays
- Filtering

Color

- 3 different cone cells in eye means color can be represented by 3 numbers (channels)
  - Cameras, displays work with Red, Green, and Blue light: RGB
- Each channel (color) represented by its own matrix “plane”
- MATLAB: `pic(row, col, ch)`
  - `pic(:,:,1)`: Red
  - `pic(:,:,2)`: Green
  - `pic(:,:,3)`: Blue



(Wikimedia Commons)



A color picture is made up of RGB matrices → 3-d array

114	114	112	112	114	111	114	115	112	113
114	113	111	109	113	111	113	115	112	113
115	114	112	111	111	112	112	111	112	112
116	117	116	114	112	115	113	112	115	114
113	112	112	112	112	110	111	113	116	115
115	115	115	115	113	111	111	113	116	114
112	113	116	117	113	112	112	113	114	113
115	116	118	118	113	112	112	113	114	114
116	116	117	117	114	114	112	112	114	115

153	153	150	150	154	151	152	153	150	151
153	152	149	147	153	151	151	153	150	151
154	153	151	150	151	152	150	149	150	150
155	156	155	152	152	155	151	150	153	153
151	150	150	150	150	148	149	151	152	151
153	153	153	153	151	149	149	151	152	150
150	151	152	153	151	150	150	151	152	151
153	154	154	154	151	150	150	151	152	152
154	154	153	153	149	149	150	150	152	153

212	212	212	212	216	213	215	216	213	213
212	211	211	209	215	213	214	216	213	213
213	212	210	209	212	214	213	212	213	212
214	215	214	214	213	216	214	213	215	212
213	212	212	212	212	210	211	213	214	211
215	215	216	216	213	211	211	213	212	210
212	213	214	215	213	212	212	213	214	213
215	216	216	216	213	212	212	213	214	214
216	216	215	215	213	213	213	213	214	215

E.g., color image data is stored in a 3-d array **A**:

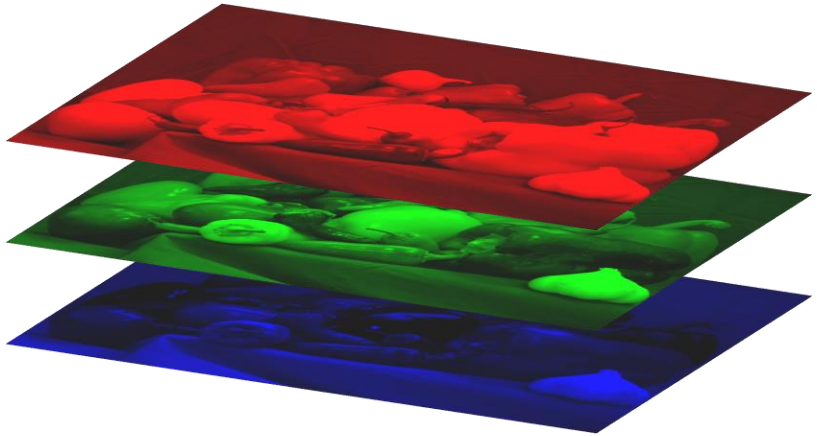
$$0 \le A(i,j,1) \le 255$$

$$0 \le A(i,j,2) \le 255$$

$$0 \le A(i,j,3) \le 255$$

Visualize a 3D array as a stack of “layers” which are 2D arrays

Color image



3-d Array

$$0 \le A(i,j,1) \le 255$$

$$0 \le A(i,j,2) \le 255$$

$$0 \le A(i,j,3) \le 255$$

Beware the two different “3”s:

- `dims = size(A)  % [720, 1280, 3]`
- `length(dims) == 3  %` A has 3 dimensions: rows, columns, layers
- `dims(3) == 3  %` A has **3** layers: red, green, blue

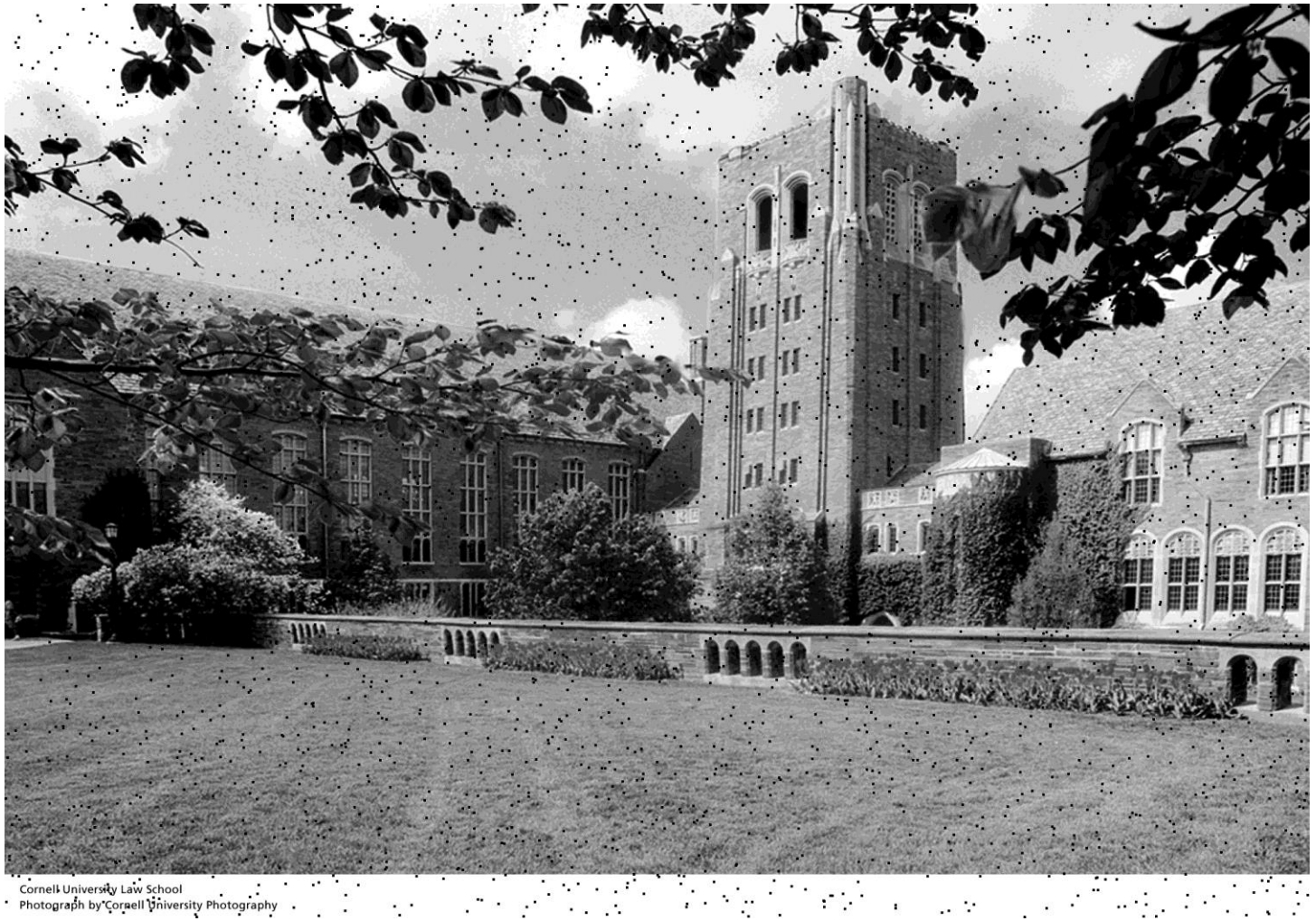
Example: Back to B&W

Replace color with “brightness”

- Average RGB value?
  - Eyes are more sensitive to green than blue
- Luminosity?
  - Requires complicated math
- Luma?
  - Good compromise
  - $Y' = 0.2126\ R' + 0.7152\ G' + 0.0722\ B'$

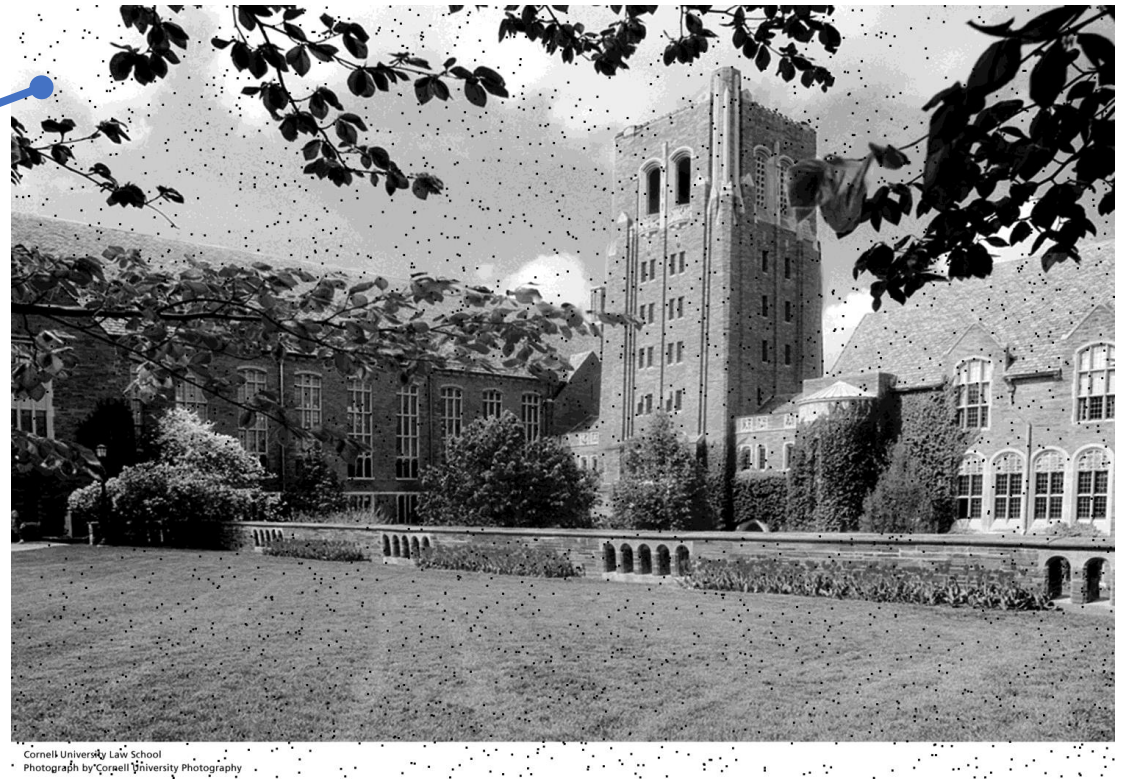
Application: median filtering

How can we remove noise?



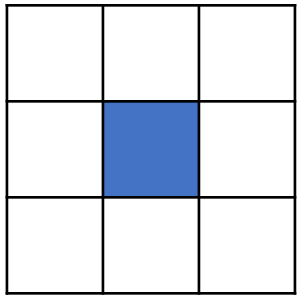
Dirty pixels look out-of-place

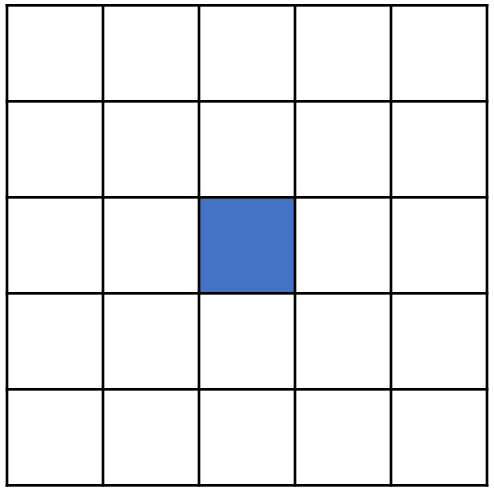
150	149	152	153	152	155
151	150	153	154	153	156
153	2	3	156	155	158
154	2	1	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162



How to fix “bad” pixels?

- Visit each pixel
- Replace with typical values from its neighborhood
  - How to choose “typical” value?
  - How big is the neighborhood?

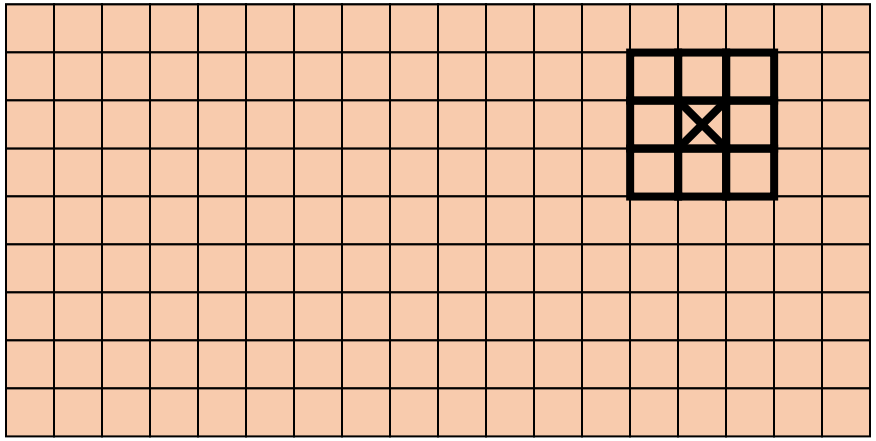




- “Typical”: mean vs. median
  - Median better for rejecting noise, preserving edges
- Neighborhood: moving window of radius *r*
  - We have seen this before! Lec 6: `minInNeighborhood`

Using a radius-1 neighborhood

0
6
6
6
6 ← median
7
7
7
7



7	7	6
7	0	6
7	6	6

Before

7	7	6
7	6	6
7	6	6

After

How to compute median

- Sort (from last lecture), then take middle element of result

or

- Call `median()` function

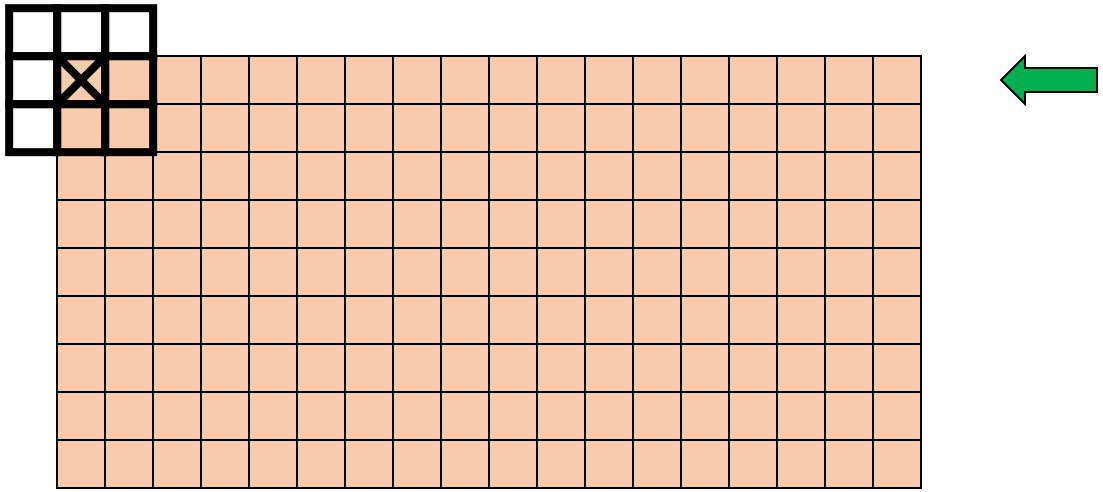
Convert to vector first!

```
v = M(:);
```

Top-down design

- Iterate over pixels
- Slice neighborhood window
- Find median value
- Replace center value with median
  - In-place or new copy?

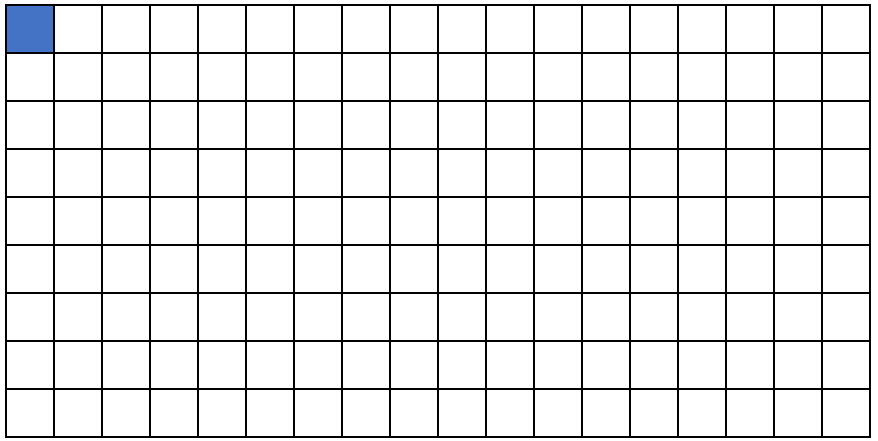
Original:



```
i = 1
```

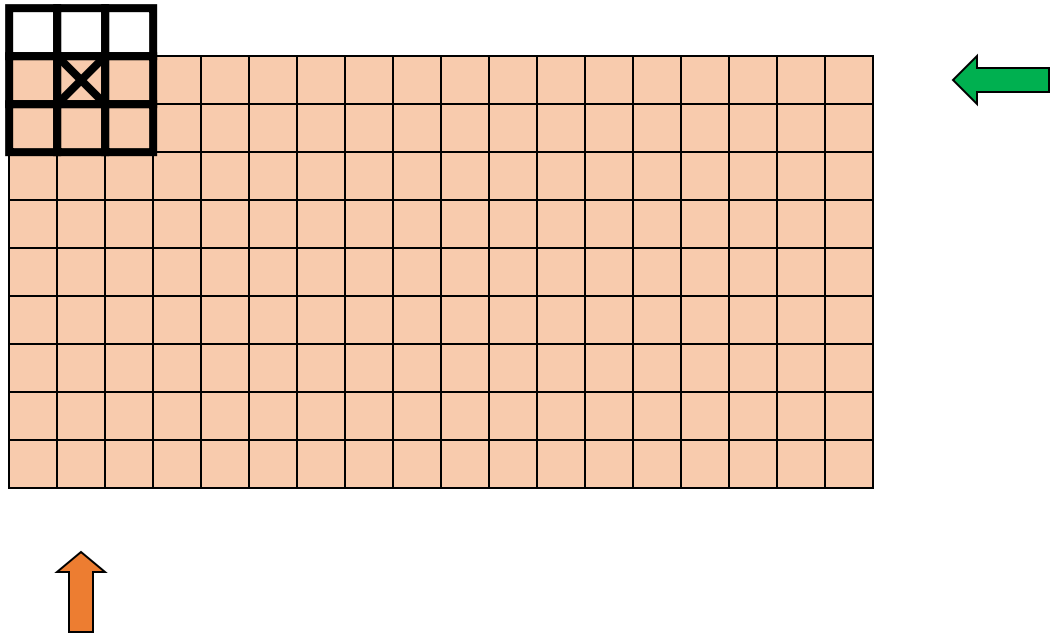
```
j = 1
```

Filtered:



Replace ⊠ with the median of the values under the window.

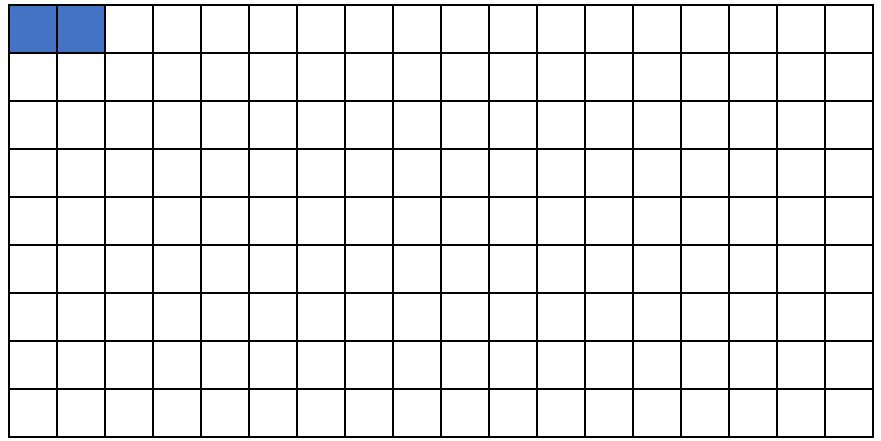
Original:



`i = 1`

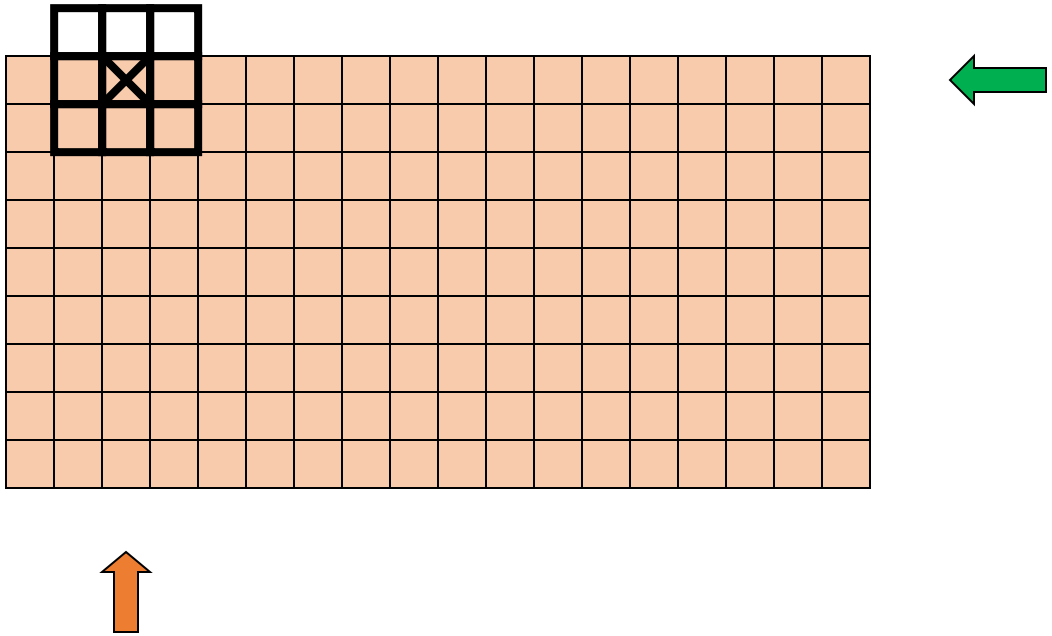
`j = 2`

Filtered:



Replace ⊠ with the median of the values under the window.

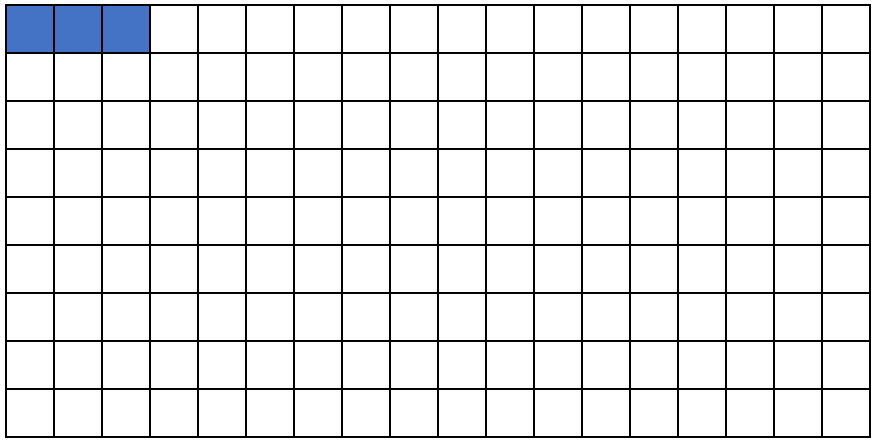
Original:



```
i = 1
```

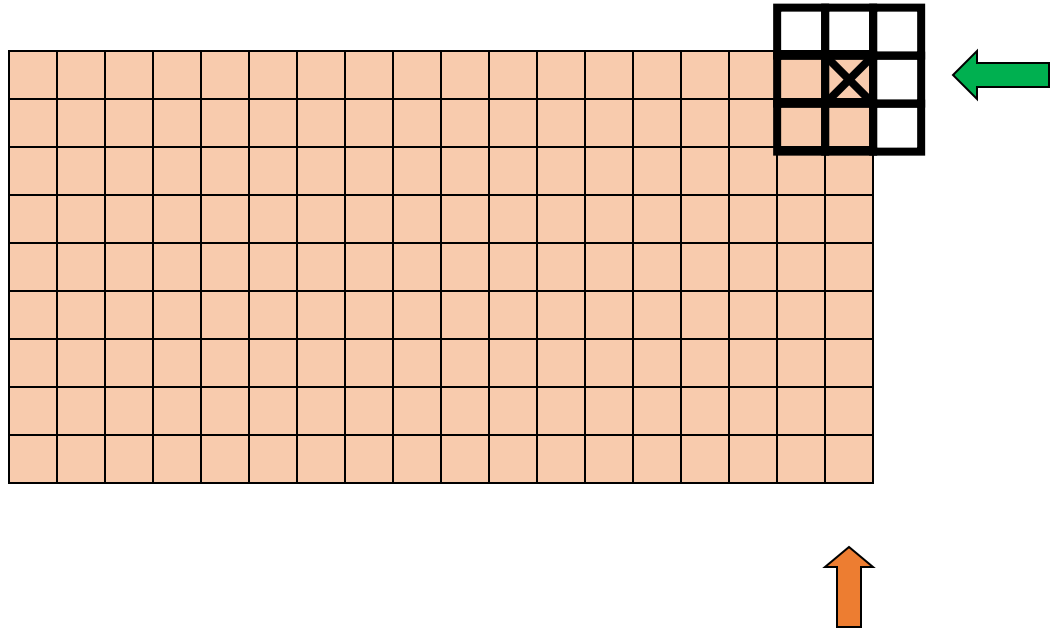
```
j = 3
```

Filtered:



Replace ⊠ with the median of the values under the window.

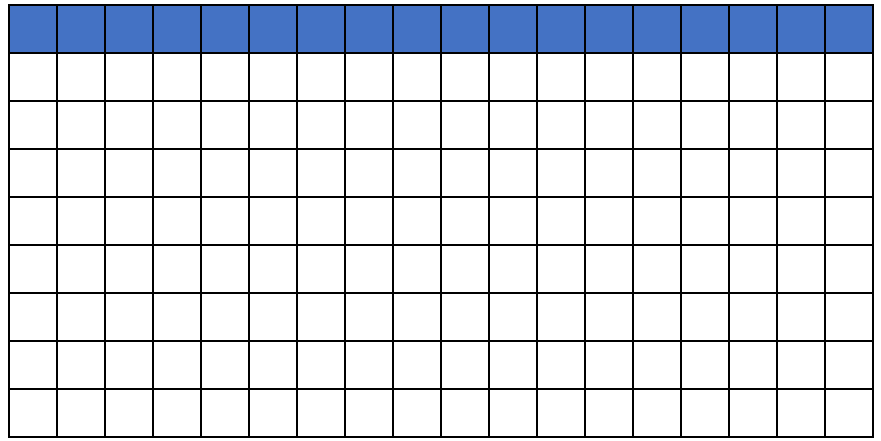
Original:



```
i = 1
```

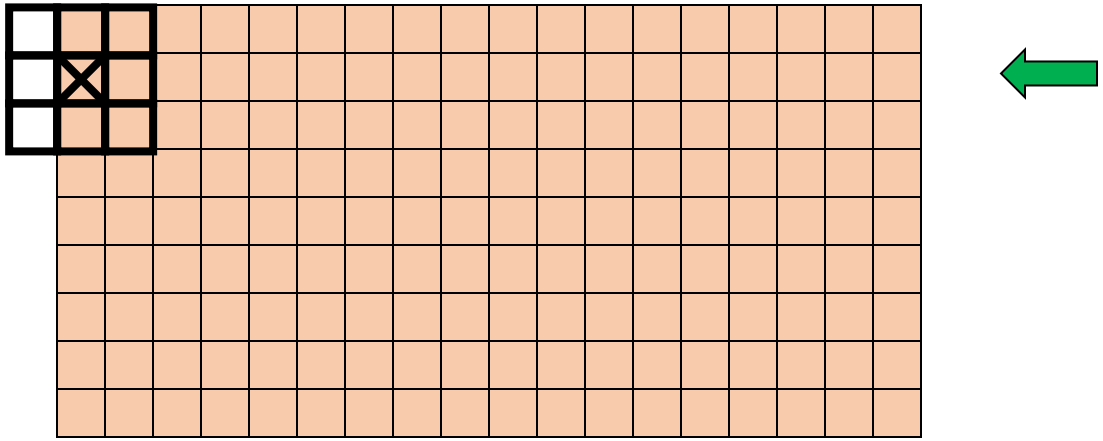
```
j = nc
```

Filtered:



Replace ⊠ with the median of the values under the window.

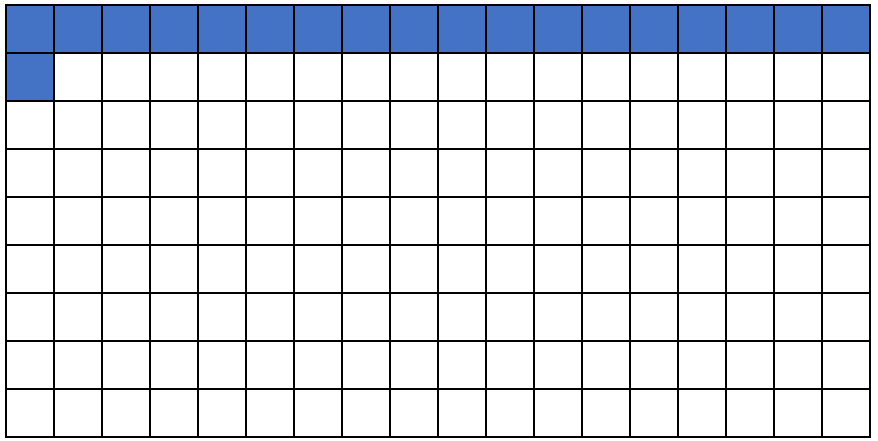
Original:



```
i = 2
```

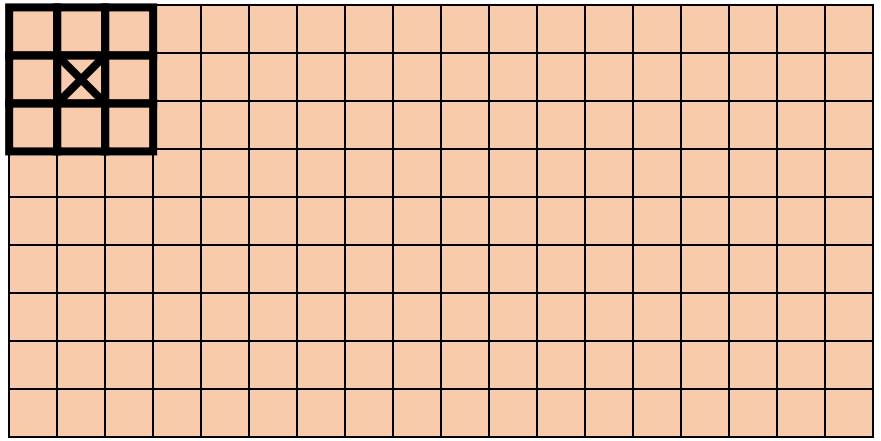
```
j = 1
```

Filtered:



Replace ⊠ with the median of the values under the window.

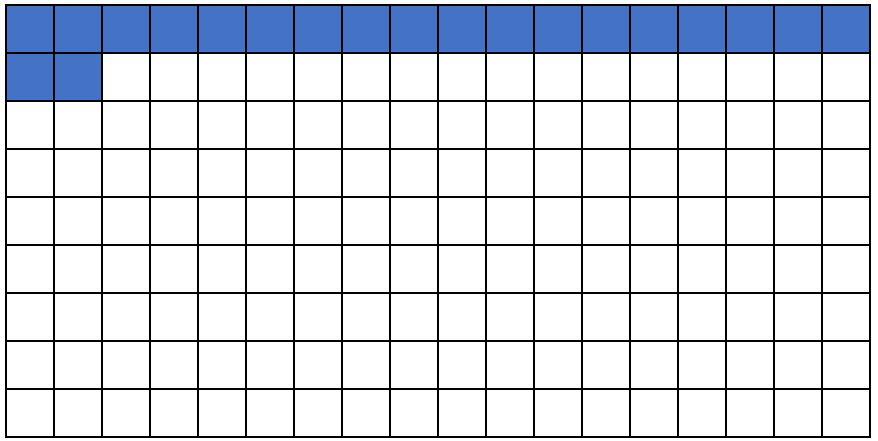
Original:



`i = 2`

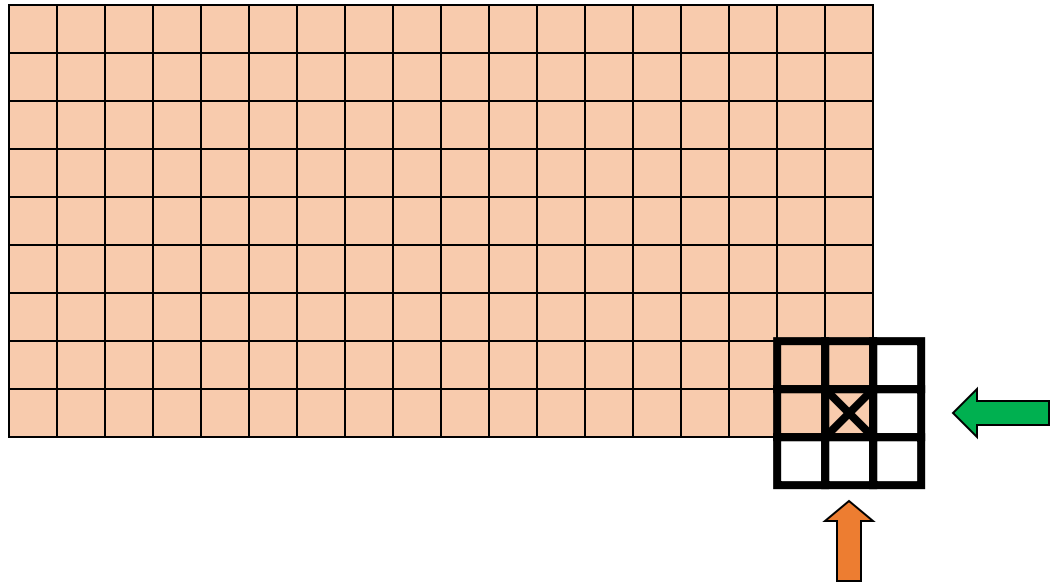
`j = 2`

Filtered:



Replace ⊠ with the median of the values under the window.

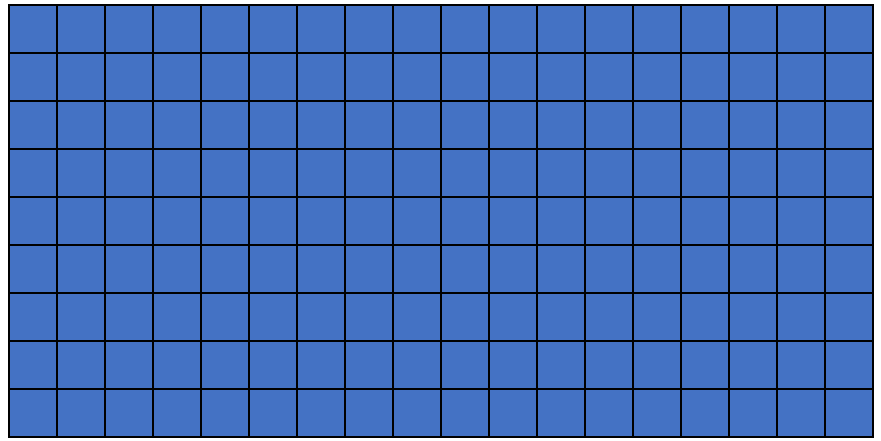
Original:



```
i = nr
```

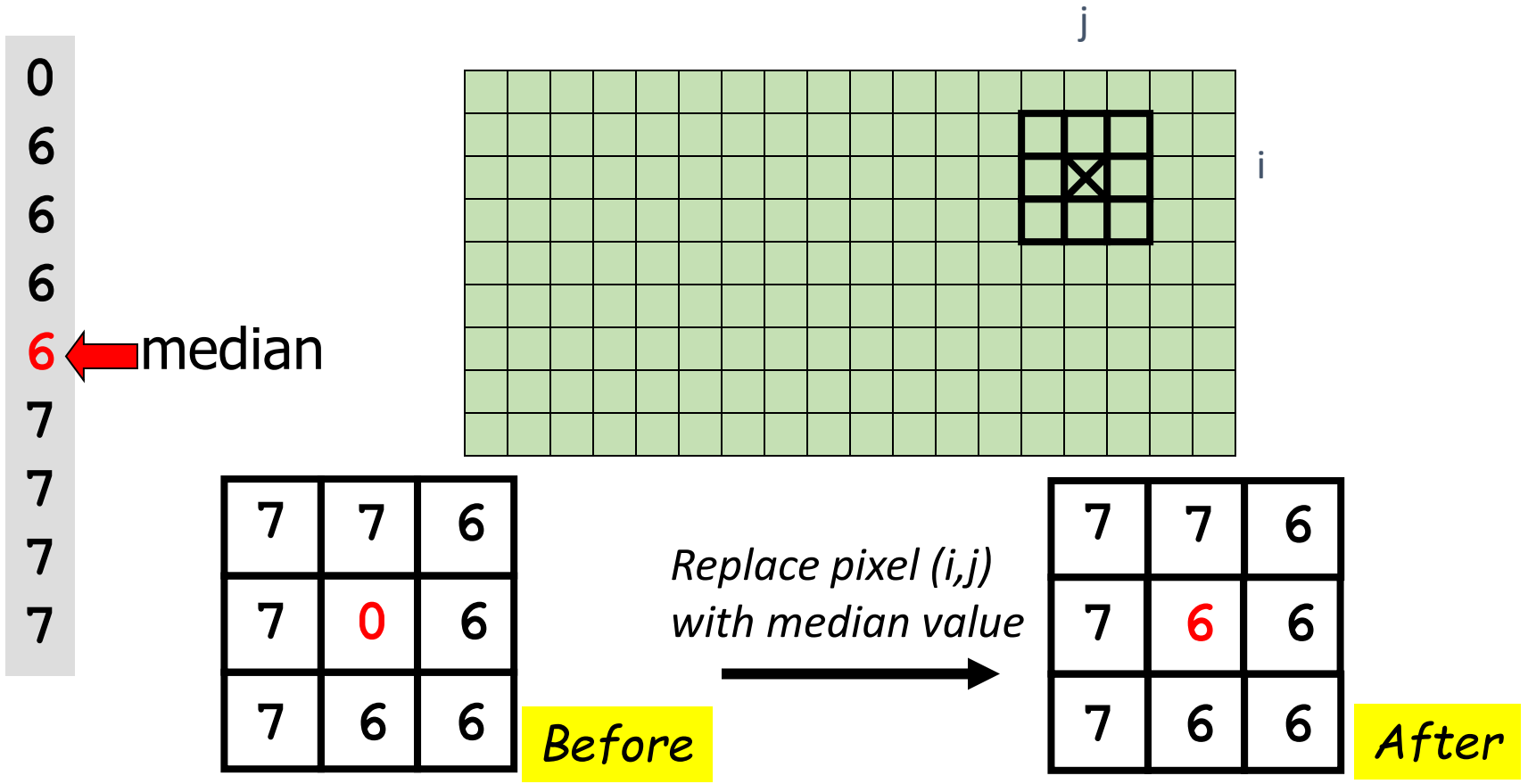
```
j = nc
```

Filtered:



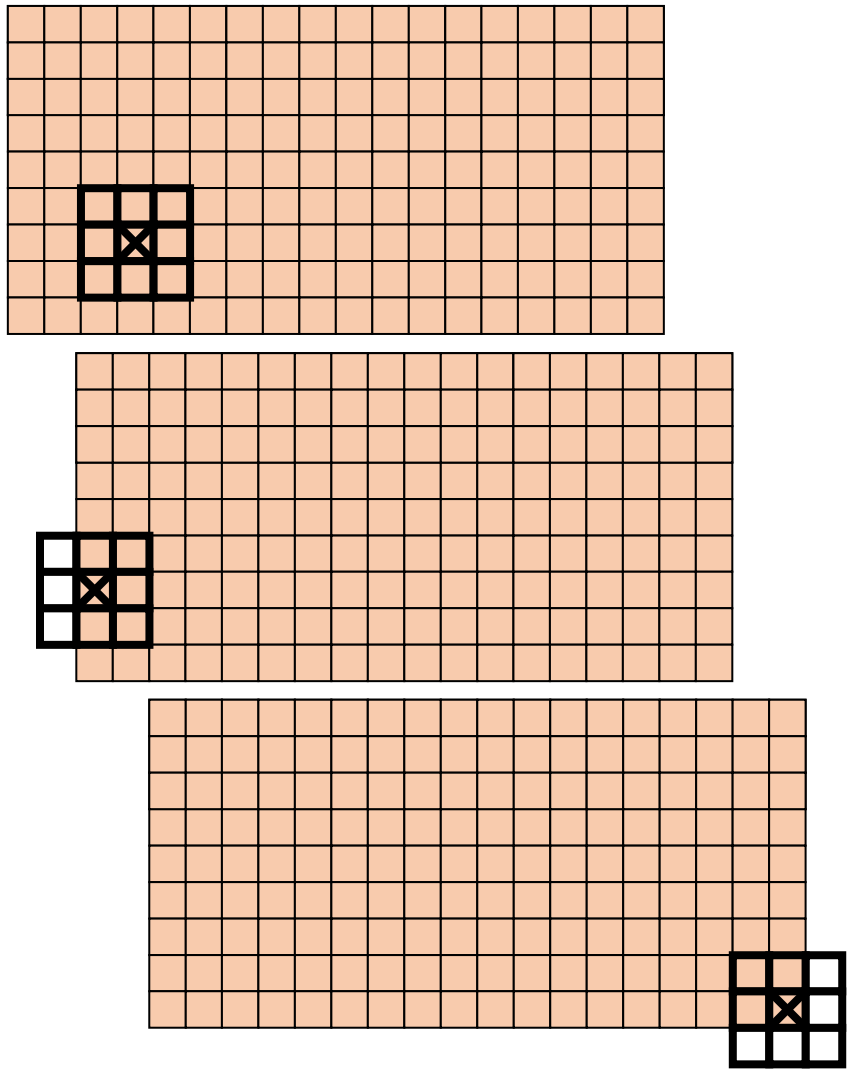
Replace ⊠ with the median of the values under the window.

Details at a pixel (i,j) with a radius 1 “neighborhood”



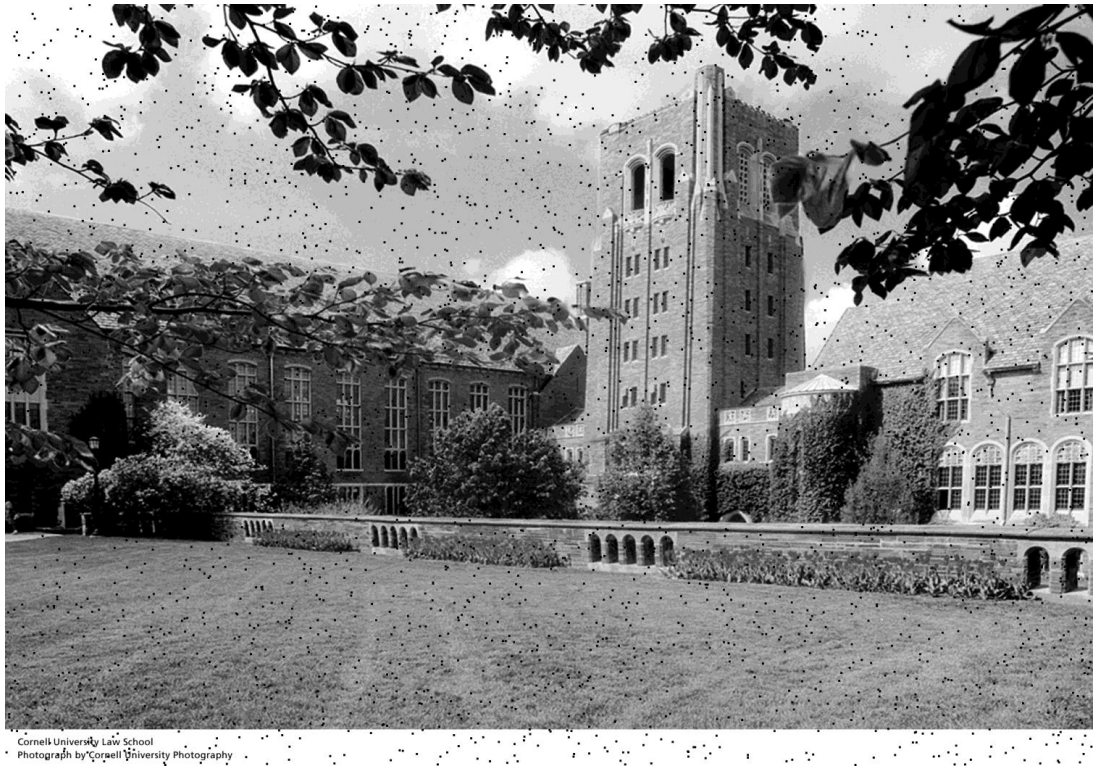

```
% Get median value in a matrix xMat
xVec= xMat(:)              % Convert matrix to vector
medianVal= median(xVec) % Use built-in function
```

Moving window: boundary issues



```
iMin= max(1,i-r)
iMax= min(m,i+r)
jMin= max(1,j-r)
jMax= min(n,j+r)
C= A(iMin:iMax,jMin:jMax)
```

Results


Cornell University Law School
Photograph by Cornell University Photography